E.U.I.T. TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA
PLAN 2000

TEMA: Redes Inalámbricas de Sensores

TÍTULO: Redes y Servicios Ubicuos Para Internet de las Cosas en Dispositivos Llevables

AUTOR: Alexandra Cuerva García

TUTOR: Dr. José Fernán Martínez Ortega **VºBº.**

DEPARTAMENTO: DIATEL

Miembros del Tribunal Calificador:

PRESIDENTE: Agustín Rodríguez Herrero

VOCAL: José Fernán Martínez Ortega

VOCAL SECRETARIO: Vicente Hernández Díaz

Fecha lectura:

Calificación: **El Secretario,**

RESUMEN DEL PROYECTO:

Este Proyecto Fin de Carrera tiene como principales objetivos: diseñar e implementar un agente de una Red Inalámbrica de Sensores (WSN) que haga uso de la tecnología Bluetooth para que se pueda comunicar tanto con la arquitectura orientada a servicios, vía radio, como con el módulo Bioharness para obtener parámetros fisiológicos; ofrecer una serie de servicios simples a la Red Inalámbrica de Sensores; diseñar un algoritmo para un sistema de alarmas; realizar e implementar una pasarela entre protocolos que usen el estándar IEEE802.15.4 (ZigBee) y el estándar IEEE802.15.1 de la Tecnología Bluetooth. Por último, implementar una aplicación Android para el reloj WiMM y que este pueda recibir alarmas en tiempo real a través del la Interfaz Bluetooth.

Este Proyecto Fin de Carrera se enmarca dentro del Proyecto Europeo de Investigación *LifeWear*, financiado por el Plan Avanza del Ministerio de Industria, Turismo y Comercio, y el Fondo Social Europeo, obteniendo el sello ITEA2, orientado a potenciar la Investigación y Desarrollo en el ámbito de los Sistemas y Servicios Software.

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA UNIVERSITARIA DE
INGENIERÍA TÉCNICA DE
TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

Redes y Servicios Ubicuos Para Internet de las Cosas en Dispositivos Llevables

Autor
Alexandra Cuerva García

Tutor
José Fernán Martínez Ortega
Dr. Ingeniero de Telecomunicación

Septiembre 2012

<< Un científico debe tomarse la libertad de plantear cualquier cuestión, de dudar de cualquier afirmación, de corregir errores. >>

Julius Robert Oppenheimer

AGRADECIMIENTOS

En primer lugar agradecer a mis padres, por el esfuerzo que han realizado durante estos años para que pudiera estudiar dicha carrera. Así como por su comprensión y confianza en mí, que ni en los malos momentos dudaron de mi capacidad para poder sacar adelante los estudios.

A mi hermana, ya que en los inicios de la carrera, me ayudó todo lo que pudo y lo que es más importante, me consoló durante los duros momentos y me ayudó a levantarme cuando las expectativas eran bajas.

A Emilio, gracias por acompañarme durante todos estos años. No solo pareja, si no también amigo y compañero. Por saber consolarme y apoyarme incondicionalmente, tanto para lo bueno como para lo malo. Tú has sido el pilar fundamental, para nunca tirar la toalla y creer siempre en mí misma. Gracias por todo.

No puedo perder esta ocasión, para agradecer a todos los compañeros que han formado parte de mi ciclo en la EUITT y que espero que sigan siéndolo. Por esas sesiones continuas en los módulos, en los que entre todos, ayudándonos, sacábamos las cosas adelante. Os llevaré siempre conmigo.

En especial, a mi amiga Luz que desde el primer día que pisé la Universidad hasta el día de hoy ha estado ahí, sin fallarme. Juntas hemos pasado muchos momentos inolvidables. Ayudándonos mutuamente, siempre que los hemos necesitado. Gracias.

A mis compañeros del departamento I + D + i, que durante estos seis meses han hecho que tanto el trabajo como la escritura de este PFC haya sido más amena. Gracias por haberme ayudado siempre que lo he necesitado.

Por último, agradecer al Dr. José Fernán Martínez, por haberme dado esta oportunidad de adentrarme en el mundo de la investigación.

RESUMEN

En las últimas décadas se han producido importantes avances tecnológicos, lo que ha producido un crecimiento importante de las Redes Inalámbricas de Sensores (RIS), conocidas en inglés como *Wireless Sensor Networks* (WSN). Estas redes están formadas por un conjunto de pequeños nodos o también, conocidos como motas, compuestos por diversos tipos de sensores.

Las Redes Inalámbricas de Sensores pueden resultar muy útiles en entornos donde el despliegue de redes cableadas, formadas por ordenadores, encaminadores u otros dispositivos de red no sea posible. Sin embargo, este tipo de redes presentan una serie de carencias o problemas que dificultan, en ocasiones, su implementación y despliegue.

Este Proyecto Fin de Carrera tiene como principales objetivos: diseñar e implementar un agente que haga uso de la tecnología Bluetooth para que se pueda comunicar tanto con la arquitectura orientada a servicios, vía radio, como con el módulo Bioharness para obtener parámetros fisiológicos; ofrecer una serie de servicios simples a la Red Inalámbrica de Sensores; diseñar un algoritmo para un sistema de alarmas; realizar e implementar una pasarela entre protocolos que usen el estándar IEEE802.15.4 (ZigBee) y el estándar IEEE802.15.1 de la Tecnología Bluetooth. Por último, implementar una aplicación Android para el reloj WiMM y que este pueda recibir alarmas en tiempo real a través del la Interfaz Bluetooth.

Para lograr estos objetivos, en primer lugar realizaremos un estudio del Estado del Arte de las Redes Inalámbricas de Sensores, con el fin de estudiar su arquitectura, el estándar Bluetooth y los dispositivos Bluetooth que se han utilizado en este Proyecto. Seguidamente, describiremos detalladamente el firmware iWRAP versión 4, centrándonos en sus modos de operación, comandos AT y posibles errores que puedan ocurrir. A continuación, se describirá la arquitectura y la especificación nSOM, para adentrarnos en la arquitectura orientada a servicios. Por último, ejecutaremos la fase de validación del sistema y se analizarán los resultados obtenidos durante la fase de pruebas.

ABSTRACT

In last decades there have been significant advances in technology, which has resulted in important growth of Wireless Sensor Networks (WSN). These networks consist of a small set of nodes, also known as spots; equipped with various types of sensors.

Wireless Sensor Networks can be very useful in environments where deployment of wired networks, formed by computers, routers or other network devices is not possible. However, these networks have a number of shortcomings or challenges to, sometimes, their implementation and deployment.

The main objectives of this Final Project are to design and implement an agent that makes use of Bluetooth technology so you can communicate with both the service-oriented architecture, via radio, as with Bioharness module for physiological parameters; offer simple services to Wireless Sensor Network, designing an algorithm for an alarm system, make and implement a gateway between protocols using the standard IEEE802.15.4 (ZigBee) and IEEE802.15.1 standard Bluetooth Technology. Finally, implement an Android application for WiMM watch that can receive real-time alerts through the Bluetooth interface.

In order to achieve these objectives, firstly we are going to carry out a study of the State of the Art in Wireless Sensor Network, where we study the architecture, the Bluetooth standard and Bluetooth devices that have been used in this project. Then, we will describe in detail the iWRAP firmware version 4, focusing on their operation modes, AT commands and errors that may occur. Therefore, we will describe the architecture and specification nSOM, to enter into the service-oriented architecture. Finally, we will execute the phase of validation of the system in a real application scenario, analyzing the results obtained during the testing phase.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

ACRÓNIMOS

ACL	Asynchronous Connection-Oriented Link
ACL-C	ACL-Control
ACL-U	ACL User Asynchronous/Isochronous
AFH	Adaptive Frequency-Hopping spread spectrum
AMP	Alternate MAC/PHY
API	Application Programming Interface
ASB	Active Slave Broadcast logical transport
ASCII	American Standard Code for Information Interchange
AVDTP	Audio/Video Distribution Transport Protocol
AVRCP	Audio/ Video Remote Control Profile
A2DP	Advanced Audio Distribution Profile
BER	bit error rate
BIP	Basic Imaging Profile
BLE	Bluetooth Low Energy
BPP	Basic Printing Profile
Bps	bauds per second
CAC	Channel Access Code
CIP	Common ISDN Acces Profile
CMOS	Complementary metal-oxide-semiconductor
CTP	Cordless Telephony Profile
CTS	Clear To Send
DAC	Device Access Code
DCD	Data Carrier Detect
DISC	Disconnect
DLC	Data Link Connection
DLCI	Data Link Connection Identifier

DM	Disconnect Mode
DPOF	Digital Print Order Format
DSR	Data Set Ready
DTR	Data Terminal Ready
DUN	Dial-up Networking Profile
ECG	Electrocardiogram
EDR	Enhaced Data Rate
EIR	Extended Inquery Response
ESB	Enterprise Service Bus
eSCO	Extended Synchronous Connection Oriented logical transport
eSCO-S	User Extended Synchronous Data logical link
FAX	FAX profile
Fcoff	Flow Control Off Command
Fcon	Flow control On Command
FCS	Frame Check Sequence
FTP	File Transfer Profile
GAVDP	General Audio/Video Distribution Profile
GFSK	Gaussian Frequency-Shift Keying
GOEP	Generic Object Exchange Profile
HCRP	Hardcopy Cable Replacement Profile
HFP	Hands-Free Profile
HID	Human Interface Device Profile
HS	High Speed
HSP	Headset Profile
HW	Hardware
IAC	Inquiry Access Code
ICP	Intercom Profile

I/O	Input/Output
I^2S	Inter-IC Sound, Integrated Interchip Sound
ISDN	Integrated Services Digital Network
ISM	Industrial, Scientific and Medical Band
JSON	JavaScript Object Notation
LAN	Local Area Network
LC	Link Control
LM	Link Manager
LMP	Link Manager Protocol
L2CAP	Logical Link Control and Adaptation Protocol
MDT	Multiplexación por División en el Tiempo
MSC	Modem Status Command
NASA	National Aeronautics and Space Administration
NSC	Non Supported Command Response
OEM	Original Equipment Manufacturer
OPP	Object Push Profile
PAN	Personal Area Networking Profile
PBAP	Phone Book Access Profile
PC	Personal Computer
PCM	Pulse Code Modulation
PN	DLC Parameter Negotation
PSB	Parked Slave Broadcast logical transport
PSK	Phase Shift Keying
RD	Received Data
RDF	Resource Description Framework
RDSI	Red Digital de servicios integrados
REJ	Reject

REST	Representation Stale Transfer
RI	Ring Indicator
RIS	Redes Inalámbricas de Sensores
RLS	Remote Line Status
RPN	Remote Port Negotation Command
RR	Receiver Ready
RSSI	Receive Signal Strength Indication
RTS	Request To Send
SABM	Set Asynchronous Balance Mode
SAR	Segmentation and reassembly
SBC	Subband Codec
SCID	Source CID
SCO	Synchronous Connection Oriented logical transport
SCO-S	User Synchronous Data logical link
SDAP	Service Discovery Profile
SDP	Session Description Protocol
SDP	Service Discovery Protocol
SIG	Bluetooth Special Interest Group
SMD	Service Mapping Description
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SOC	Service-oriented computing
SPDIF	Sony/Philips Digital Interconnect Format
SPP	Serial Port Profile
SSP	Secure Simple Pairing
SYNC	Synchronization Profile
TD	Transmit Data

UA	Unnumbered Acknowledgement
UART	Universal Asynchronous Receiver-Transmitter
UIH	Unnumbered Information with Header Check
URL	Uniform Resource Locator
USB	Universal Serial Bus
UUID	Universally Unique Identifier
VDP	Video DIstribution Profile
VM	Virtual Machine
Wi-Fi	Wireless Fidelity
WPAN	Wireless Personal Area Network
WSDL	Web Services Description Language
WSN	Wireless Sensor Network
8DPSK	8 Differencial Phase-Shift Keying
π/4-DQPSK	π/4 Differencial Quadrature Phase-Shift Keying

Capítulo 1

Introducción y objetivos

1.1 INTRODUCCIÓN A LAS REDES INALÁMBRICAS DE SENSORES

En las últimas décadas se han producido importantes avances tecnológicos, lo que ha producido un crecimiento importante de las redes inalámbricas. Gracias a la implementación de circuitos electrónicos cada vez más pequeños, con bajo consumo de energía y de menos coste ha propiciado la aparición de pequeños sensores inalámbricos. Estos sensores forman las Redes Inalámbricas de Sensores (RIS), conocidas en inglés como *Wireless Sensor Networks* (WSN). Estas redes juegan un papel muy importante en los llamados espacios inteligentes, ya que se encargan de la percepción, captación, tratamiento y distribución de la información que se obtiene a partir de cualquier parámetro ambiental.

Estas Redes Inalámbricas de Sensores, están formadas por un conjunto de pequeños nodos o también, conocidos como motas, compuestos por diversos tipos de sensores que controlan múltiples parámetros ambientales, como pueden ser la temperatura, la humedad o la luminosidad. Estos nodos autónomos pueden comunicarse entre sí vía radio. De modo que podemos referirnos a estas redes, como redes ad-hoc y ubicuas, con capacidad de transmitir información, de manera inalámbrica, y que pueden ofrecer una serie de servicios en el escenario donde se desplieguen.

Las Redes Inalámbricas de Sensores pueden resultar muy útiles en entornos donde el despliegue de redes cableadas, formadas por ordenadores, encaminadores u otros dispositivos de red no sea posible. Por esto, actualmente, las RIS están teniendo diversas aplicaciones en los campos de la automoción, de la vigilancia, en espacios industriales, en entornos inteligentes, monitorizando el medio ambiente y en un sinfín de nuevas aplicaciones.

Sin embargo, este tipo de redes presentan una serie de carencias o problemas que dificultan, en ocasiones, su implementación y despliegue. Los nodos [Hernando&08] poseen un suministro de energía muy limitado, ya que están alimentados por pequeñas baterías. Lo que conlleva a que tengan un corto tiempo de vida y por tanto a que pierdan su funcionalidad. Además, su capacidad de memoria y procesado también están limitados, debido a que son dispositivos de capacidad limitada, baja velocidad y corto alcance. Otro problema importante viene dado por la heterogeneidad de los nodos, ya que existen diversos fabricantes y en ocasiones los modelos son incompatibles entre sí.

Por lo mencionado anteriormente, existe la necesidad de la utilización de herramientas y mecanismos que abstraigan a los programadores y desarrolladores de los niveles más bajos de programación.

Es por todo ello, que el desarrollo de aplicaciones para estas redes ad-hoc, así como la implementación de protocolos y arquitecturas de red, se ha convertido en un campo de investigación muy importante en estos últimos años.

1.2 OBJETIVOS DEL PROYECTO

Esta memoria se centrará en los aspectos relacionados con el diseño y desarrollo de aplicaciones que se adapten al entorno de las Redes Inalámbricas de Sensores, en concreto a lo que concierne a la tecnología Bluetooth. Podemos destacar los siguientes objetivos principales en este Proyecto Fin de Carrera:

- Diseñar e implementar un agente que haga uso de la tecnología Bluetooth para que se pueda comunicar tanto con la arquitectura orientada a servicios, vía radio, como con el módulo Bioharness para obtener parámetros fisiológicos.

- Ofrecer una serie de servicios simples a la Red Inalámbrica de Sensores, para que dicha red, a partir de estos servicios, pueda componer servicios más complejos.

- Diseñar un algoritmo para un sistema de alarmas. A partir de los parámetros obtenidos del módulo Bioharness y de otros parámetros de la arquitectura de red, como por ejemplo, la temperatura ambiental, generar una serie de alarmas que permitan al deportista saber si corre cierto riesgo al practicar deporte o no.

- Realizar e implementar una pasarela entre protocolos que usen el estándar IEEE802.15.4 (ZigBee) y el estándar IEEE802.15.1 de la Tecnología Bluetooth. Es decir, tendremos un nodo equipado con una interfaz Bluetooth que se encargará de recibir alarmas vía radio y la transmitirá a través de dicha interfaz Bluetooth a un reloj.

- Implementar una aplicación Android para el reloj WiMM y que este pueda recibir alarmas en tiempo real a través del la Interfaz Bluetooth. Así, el deportista, que irá equipado con dicho reloj, estará informado en todo momento de si está en riesgo o no.

1.3 ORGANIZACIÓN DE CONTENIDOS

- **Capítulo 1:** en este capítulo se hace una pequeña introducción a las Redes Inalámbricas de Sensores. A continuación, se analizan los objetivos principales del Proyecto y el marco en el que ha sido desarrollado.
- **Capítulo 2:** este capítulo es el más extenso de todos. Se abarca el Estado del Arte en las Redes Inalámbricas de Sensores. Se describen su arquitectura y los componentes que lo forman. A continuación, se estudiará en profundidad el estándar Bluetooth, donde se analizarán sus orígenes e historia, el alcance, las topologías de red que permite y las especificaciones. Así como, los perfiles Bluetooth en general y en el perfil SPP en particular. Por último, se describirán los dispositivos Bluetooth utilizados en este Proyecto.
- **Capítulo 3:** este capítulo está dedicado al firmware iWRAP v4. Primeramente, se realiza una pequeña introducción. A continuación, se describen los modos de operación y los comandos AT. También se abarcan los posibles mensajes de error SDP y RFCOMM que pueden ocurrir durante una conexión Bluetooth. Por último, se describen dos ejemplos de conexión Bluetooth empleando el perfil SPP.
- **Capítulo 4:** en este capítulo se aborda todo lo relacionado con la arquitectura y especificación nSOM. Comenzamos con una pequeña introducción, para adentrarnos en la arquitectura orientada a servicios y, en concreto, en nuestra arquitectura nSOM. Más adelante, se explican los agentes y los subsistemas que forman el nSOM. A continuación, se describe el modelo de comunicación, poniendo ejemplos tanto de la exposición de servicios simples como de servicios compuestos. Por último, se aborda el diseño detallado de la arquitectura, centrándonos básicamente en los sistemas Bluetooth.
- **Capítulo 5:** este capítulo trata de la validación del sistema y el análisis de los resultados. Comenzamos con una introducción, para continuar con la justificación del escenario planteado. Posteriormente se estudian los casos de uso de dicho escenario, centrándonos en las infraestructuras de las motas ZephyrBT y AlarmasBT. A continuación, se describe el escenario de la validación. Por último, se analizan los resultados de la validación obtenidos.
- **Capítulo 6:** este capítulo se centra en las conclusiones de este Proyecto Fin de Carrera, realizando un breve resumen de los contenidos que se han abordado en cada capítulo. Por último, se habla de las mejoras que se pueden realizar y de los trabajos futuros que pueden ayudar a optimizar el sistema.

1.4 MARCO DE DESARROLLO DEL PROYECTO FIN DE CARRERA

Este Proyecto Fin de Carrera se enmarca dentro del Proyecto Europeo de Investigación *LifeWear*, financiado por el Plan Avanza del Ministerio de Industria, Turismo y Comercio, y el Fondo Social Europeo, obteniendo el sello ITEA2, orientado a potenciar la Investigación y Desarrollo en el ámbito de los Sistemas y Servicios Software [LifeWear 12].

Capítulo 2

Estado del Arte en Redes Inalámbricas de Sensores

2.1 ARQUITECTURA Y COMPONENTES

Una red de sensores inalámbricos [Fernández&09] [Hernando&08] está formada por un conjunto de dispositivos autónomos, denominados nodos o motas, los cuales están equipados con una serie de sensores. Tienen capacidad de comunicación inalámbrica, de manera, que forman redes ad-hoc sin administración central ni infraestructura física preestablecida.

En la Figura 1 se puede observar la arquitectura de una Red de Sensores Inalámbricos, es una red compleja que adquiere y distribuye datos. A su vez, está monitorizada y controlada por un centro de control.

La arquitectura física de una RIS está formada por un conjunto de motas [Augusto&], un punto de acceso central conocido como Gateway, puerta de enlace o sumidero y por último, una estación base. El número de sensores y el despliegue con una densidad espacial de los mismos vendrá determinado por la aplicación a la que esté orientada.

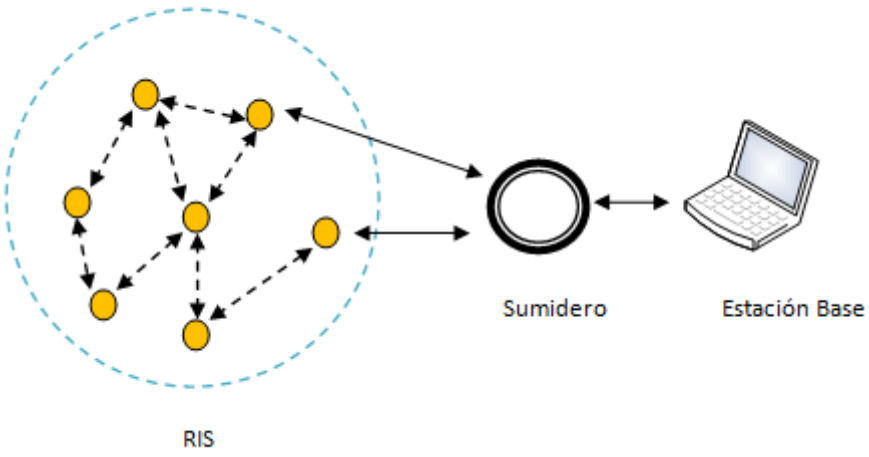


Figura 1: Arquitectura básica de una Red Inalámbrica de Sensores.

2.1.1 Nodos

Los nodos, también llamados motas, son dispositivos electrónicos que pueden realizar algún tipo de procesamiento, recopilar información a través de sus sensores y comunicarse con otros nodos que pertenecen a la red. El hardware básico [Karl&07] de un nodo sensor se compone de un transceptor (transmisor/receptor), un procesador, uno o más sensores, memoria y batería o fuente de alimentación, como se observa en la Figura 2.

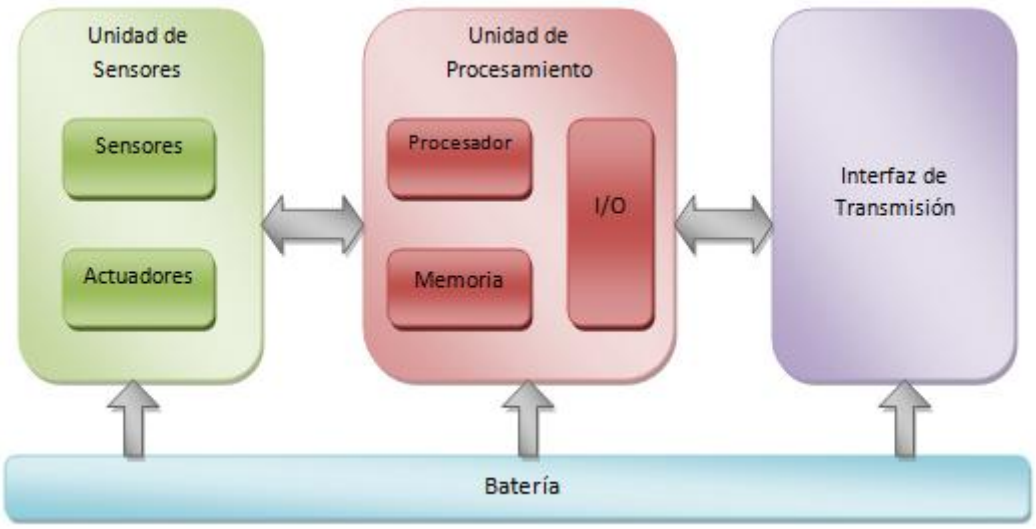


Figura 2: Arquitectura general de un nodo.

Una mota puede disponer de uno o más sensores. Un sensor es un transductor que es capaz de detectar magnitudes físicas o químicas y transformarlas en señales eléctricas que se pueden medir mediante procesados digitales.

2.1.2 Puerta de Enlace

La puerta de enlace, también conocida como nodo sumidero, o Gateway en inglés. Es un nodo que no está equipado con la unidad de sensores, y cuyo objetivo principal es actuar como pasarela entre dos redes de diferente tipo. Recogiendo, así, los datos de la RIS e interconectando esta red con el mundo exterior, por ejemplo, a otra red de área local LAN.

2.1.3 Estación Base

La estación base, es otro elemento que forma parte de la arquitectura de un Red Inalámbrica de Sensores. Se encarga de recoger los datos adquiridos por la RIS a través de la puerta de enlace. Se corresponde con un ordenador, donde todos los datos recogidos van a parar a una base de datos. Desde la cual, los usuarios pueden solicitar toda la información, de manera local o remota, para su posterior estudio o análisis.

2.2 ESTÁNDAR BLUETOOTH

El estándar Bluetooth es una especificación industrial para Redes Inalámbricas de Área Personal (WPAN) que posibilita la transmisión de voz y datos entre diferentes dispositivos mediante un enlace por radiofrecuencia en la banda científica y médica (ISM) de los 2,4 GHz a los 2,485 GHz. Emplea el espectro ensanchado por salto de frecuencia y una señal full-dúplex con una tasa nominal de 1600 saltos/s. Se presenta como la tecnología inalámbrica ideal para la conexión de dispositivos electrónicos.

Los principales objetivos que pretende conseguir esta tecnología son los siguientes:

- Facilitar las comunicaciones entre dispositivos móviles y fijos.
- Eliminar cables y conectores entre éstos.
- Ofrecer la posibilidad de crear pequeñas redes inalámbricas y facilitar la sincronización y transmisión de datos entre equipos.

En la actualidad, existen un número de dispositivos muy amplio que incorpora la tecnología Bluetooth. Los principales dispositivos pertenecen al ámbito de las telecomunicaciones y la informática personal, como PDAs, teléfonos móviles, ordenadores personales y portátiles, impresoras o manos libres.

2.2.1 Orígenes

El nombre Bluetooth proviene del rey danés y noruego Harald Blatand, ya que su traducción al inglés sería Harold Bluetooth. Era conocido por ser buen comunicador y por unificar las tribus noruegas, suecas y danesas.

Su símbolo es la unión de las runas nórdicas análogas a las letras H y B:

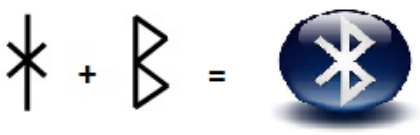


Figura 3: Origen del logo Bluetooth.

2.2.2 Historia

[Bluetooth 12] En el año 1994, la compañía sueca Ericsson comenzó un estudio para estudiar la viabilidad de una interfaz radio, de bajo coste y consumo, para la interconexión de teléfonos móviles y periféricos con el objetivo de eliminar el cableado.

A principios del año 1997, otros fabricantes despertaron su interés por este proyecto. Por lo que en 1998 se creó un Grupo de Especial Interés Bluetooth (SIG) [Bluetooth SIG 12], formado por las empresas Ericsson, Nokia, IBM, Toshiba e Intel. Su principal objetivo era lograr un conjunto de áreas de negocio para explotar dicha tecnología.

Posteriormente, en diciembre de 1999, se unieron 3Com, Lucent, Microsoft y Motorola como promotores del SIG.

Poco a poco, se fueron uniendo compañías de diversos ámbitos, desde las telecomunicaciones y la informática hasta la industria musical y textil. En este año 2012, ya son más de 16.000 las compañías pertenecientes al SIG.

Figura 4: Logo de la tecnología Bluetooth [Bluetooth 12].

2.2.3 Alcance

Los dispositivos que implementan la tecnología Bluetooth pueden comunicarse entre ellos siempre que se encuentren dentro de su alcance. Las comunicaciones se realizan vía radio, de forma que los dispositivos no tienen que estar alineados y pueden incluso estar en habitaciones separadas, siempre que la potencia de transmisión tenga un valor adecuado y lo permita. Los dispositivos Bluetooth pueden clasificarse en tres clases (Clase 1, Clase 2 y Clase 3) conforme a la potencia emitida. A mayor potencia, más rango de alcance.

Los dispositivos de diferentes clases son totalmente compatibles entre sí [Bluetooth Especificaciones 12]. En la siguiente tabla se pueden observar la potencia máxima de transmisión permitida y el alcance aproximado de cada una de las clases de dispositivos Bluetooth.

CLASE	POTENCIA DE TRANSMISIÓN MÁX. PERMITIDA	ALCANCE (aproximado)
Clase 1	20 dBm	~ 100 m
Clase 2	4 dBm	~ 10 m
Clase 3	0 dBm	~ 1 m

Tabla 1: Clasificación dispositivos Bluetooth conforme a la potencia máxima de transmisión.

La clase 2 es la más empleada en los teléfonos móviles. La clase 3 se encuentra en dispositivos para el uso industrial.

2.2.4 Topologías de red

La tecnología Bluetooth [Hopkins&03] puede proporcionar conexiones punto a punto y conexiones punto a multipunto. Dos o más dispositivos que comparten el mismo canal físico forman una red ad-hoc, llamada *piconet*. Uno de los dispositivos toma el rol de maestro de la *piconet*, mientras que el resto actúan como esclavos. La especificación Bluetooth permite que cualquier equipo asuma cualquiera de los dos papeles, incluso un equipo puede actuar como maestro en un enlace y como esclavo en otro. El maestro de la *piconet* es el encargado de la sincronización, estableciendo el patrón de saltos en base a su dirección y reloj.

El número máximo de dispositivos por *piconets* es de ocho. Como uno asume el rol de maestro, como máximo puede haber siete equipos esclavos activos. La *piconet* puede tener muchos más dispositivos esclavos, eso sí, en estado de espera.

Múltiples *piconets* con áreas de cobertura solapadas forman una *scatternet*. Cada *piconet* solo puede tener un maestro. Sin embargo, los esclavos pueden participar en *piconets* diferentes utilizando multiplexación por división en el tiempo (MDT). También puede darse el caso de que un mismo dispositivo opere como maestro y como esclavo simultáneamente en distintas *piconets*.

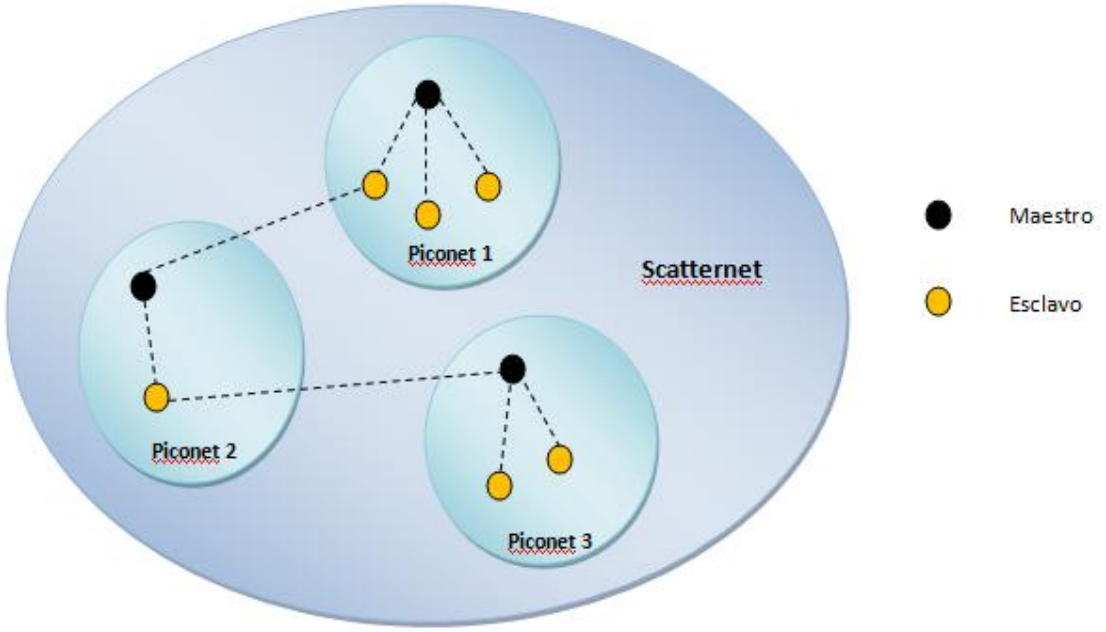


Figura 5: Ejemplos de redes piconet y scatternet.

Dependiendo de la información que tenga el maestro sobre el dispositivo esclavo al que quiere conectarse se siguen dos procedimientos diferentes:

- Si no se tiene ningún tipo de información sobre el dispositivo remoto, lo primero que se debe hacer es un procedimiento de búsqueda, denominado *inquiry* y otro de emparejamiento, llamado *page*.

- Si se conocen detalles sobre el dispositivo al que se quiere conectar, solo se realizará el procedimiento de emparejamiento *page*.

El procedimiento de búsqueda de dispositivos, *inquiry*, permite a un dispositivo Bluetooth descubrir otros dispositivos que se encuentran en su área de cobertura y así, conseguir la información necesaria para conectarse con ellos. Posteriormente, el procedimiento de asociación, *page*, permite el establecimiento de la conexión una vez conocida la dirección del dispositivo remoto.

Una vez hecho el descubrimiento, habitualmente, los dispositivos Bluetooth recuerdan a los dispositivos con los que se han conectado previamente. De esta manera, en la próxima conexión no será necesario realizar estos pasos previos.

2.2.5 Especificaciones Bluetooth

Desde 1994 han ido apareciendo diferente versiones [Bluetooth Especificaciones 12] con importantes mejoras, sobre todo con lo que respecta al consumo de energía, velocidades de transferencia y seguridad. Todas ellas están diseñadas para la compatibilidad hacia abajo, es decir, el último estándar cubre a todas las versiones anteriores.

2.2.5.1 Bluetooth v1.0 y v1.0b

Las versiones 1.0 y 1.0b han tenido muchos problemas, siendo el más grave el de interoperabilidad entre productos de diferentes fabricantes. En estas versiones se incluye a nivel hardware, de forma obligatoria, la dirección del dispositivo Bluetooth en la transmisión lo que hace imposible el anonimato a nivel de protocolo.

2.2.5.2 Bluetooth v1.1

Está ratificado como estándar IEEE 802.15.1-2002. Se corrigieron muchos errores con respecto a la versión anterior. Con respecto a la seguridad, se añadió soporte para canales no encriptados. Se añade un nuevo indicador, el indicador de señal de potencia recibida (RSSI).

2.2.5.3 Bluetooth v1.2

Esta versión tiene importantes mejoras con respecto a versiones anteriores. En concreto, se consigue una conexión más rápida y se incorpora el mecanismo de "Discovery" (*inquiry*). Se mejora la resistencia frente a interferencias radio, empleando el salto de frecuencia adaptable de espectro ampliable (AFH). Se consigue mayor velocidad de transmisión, hasta 721 kbit/s. También se mejora la calidad de la voz en los enlaces de audio ya que se permite la retransmisión de paquetes gracias al radio enlace eSCO y se incorpora el control de flujo y los modos de retransmisión L2CAP entre otras. Esta versión está ratificada como estándar IEEE 802.15.1-2005.

2.2.5.4 Bluetooth v2.0 + EDR

Esta versión de la especificación es la principal del estándar Bluetooth y fue lanzada en 2004. Por supuesto, es compatible con la versión anterior. La mejora más importante que se introduce es una velocidad de datos mejorada (EDR) para acelerar la transferencia de datos. La tasa nominal es de 3Mb/s, aunque en la práctica se consiguen hasta 2,1 Mb/s. EDR utiliza una

combinación de modulaciones GFSK y PSK, con dos variantes π/4-DQPSK 8DPSK. Otra ventaja, es el menor consumo de energía a través de un ciclo de trabajo reducido.

2.2.5.5 Bluetooth v2.1 + EDR

Fue adoptada por el Bluetooth SIG en julio de 2007 [Specification 2.1 + EDR 07]. La principal función que incorpora es SSP lo que conlleva a la mejora de la experiencia de emparejamiento de dispositivos Bluetooth. Además incluye la "respuesta de investigación extendida" (EIR) que proporciona más información durante el procedimiento de descubrimiento de dispositivos antes de la conexión para mejorar el filtrado. Lo que reduce el consumo de energía hasta en cinco veces con respecto a versiones anteriores.

2.2.5.6 Bluetooth v3.0 + HS

A mediados de 2009, aparece una nueva versión que incorpora el soporte de velocidades de transferencia de datos de hasta 24 Mb/s, aunque no a través del enlace Bluetooth. En realidad, la conexión Bluetooth se emplea para establecer la conexión. Si la transferencia de datos que se debe realizar es elevada, se emplea AMP, es decir, se envía empleando el estándar IEEE 802.11, principalmente usando la tecnología Wi-Fi.

2.2.5.7 Bluetooth v4.0

Es la última versión Bluetooth y fue adoptada en junio de 2010. Incluye el Bluetooth clásico, la alta velocidad (HS) y protocolos Bluetooth de bajo consumo. Cabe destacar la incorporación del Bluetooth Low Energy (BLE), es un subconjunto Bluetooth con una pila de protocolos completamente nuevo para conseguir una tecnología de bajo consumo.

El protocolo empleado en el BLE no es compatible con el que usa el Bluetooth clásico. Un dispositivo (dispositivo Bluetooth modo dual) puede operar tanto con BLE como con el Bluetooth clásico, pero no de manera simultánea.

2.2.6 Perfiles Bluetooth

Un perfil Bluetooth [Bluetooth Perfiles 12] define una interfaz de alto nivel para emplearla entre dispositivos Bluetooth. Un mismo dispositivo es capaz de soportar varios perfiles. En concreto, define una selección de mensajes y protocolos para implementar una aplicación determinada. Una especificación de perfil, debe contener una información mínima:

- Dependencia con respecto a otros perfiles.
- Sugerencias acerca del formato de las interfaces de usuario.
- Partes específicas de la pila Bluetooth que se emplean. Como son los protocolos, opciones particulares y parámetros en cada capa de la pila.

El SIG Bluetooth ha adoptado un gran número de perfiles que muestran diferentes tipos de escenarios y unas características determinadas donde pueden ser empleados. La siguiente figura muestras los perfiles Bluetooth que existen:

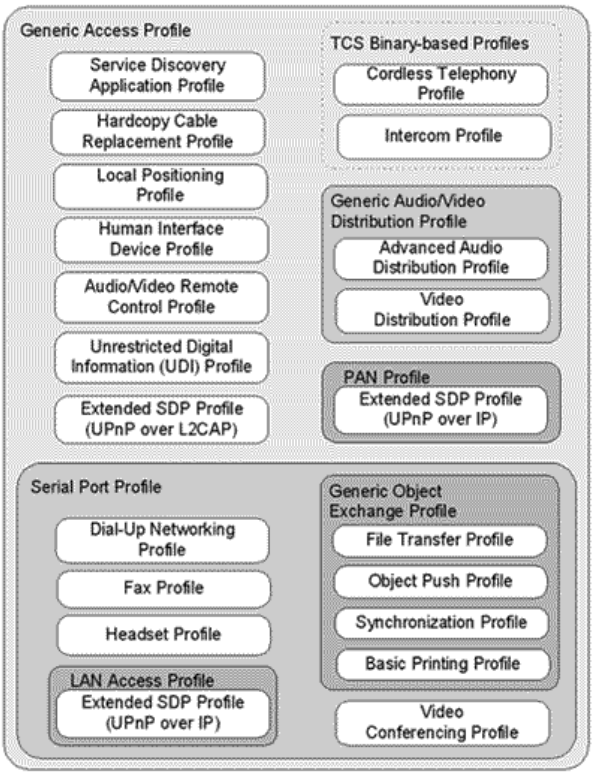


Figura 6: Estructura de perfiles Bluetooth [Bluetooth Perfiles 12].

2.2.6.1 Perfil de distribución de audio avanzada (A2DP)

Este perfil define como propagar un stream de audio de alta calidad (mono o estéreo) entre dispositivos Bluetooth en canales ACL. Este perfil utiliza el protocolo AVDTP y el perfil GAVDP. Incluye soporte para códecs sub-banda de baja complejidad (SBC) y soporta, opcionalmente, MPEG-1, MPEG-2, AAC y ATRAC.

2.2.6.2 Perfil de control remoto de audio y video (AVRCP)

Está diseñado para ofrecer una interfaz estándar que sea capaz de controlar dispositivos, como pueden ser televisores y equipos de música, entre otros. De esta manera, conseguiríamos que un mando único fuera capaz de controlarlo todo.

2.2.6.3 Perfil de tratamiento básico de imágenes (BIP)

Este perfil se emplea para el envío de imágenes, incluyendo capacidades de ajuste de tamaño y conversión de formatos. Se pueden distinguir las siguientes funciones dentro de este perfil:

- "*Image Push*": permite el envío de imágenes desde un dispositivo controlado por el usuario.
- "*Image Pull*": permite la recepción de imágenes desde un dispositivo remoto.
- "*Advanced Image Printing*": permite la impresión de imágenes con opciones avanzadas usando el formato DPOF.
- "*Automatic Archive*": habilita backups automáticos de todas las imágenes de un dispositivo remoto.
- "*Remote Camera*": permite el uso de una cámara digital de manera remota.
- "*Remote Display*": su objetivo es el envío de imágenes a otro dispositivo para que se puedan visualizar.

2.2.6.4 Perfil de impresión básica (BPP)

Es un perfil que permite el envío de texto, emails, vCards, imágenes y otros documentos a impresoras en base a trabajos de impresión. BPP se diferencia de HCRP en que no se necesitan drivers específicos de impresora, lo que lo convierte en un perfil muy apropiado para dispositivos móviles, aislándolos de dicha problemática.

2.2.6.5 Perfil de acceso común a RDSI (CIP)

Sus propósitos son:

- Definir como las aplicaciones deben accede a los servicios RDSI ó ISDN (en inglés) vía Bluetooth.
- Permitir, siempre que sea posible, acceso ilimitado a los servicios, datos o señalización proporcionada por RDSI.

- Garantizar que las aplicaciones RDSI heredadas siguen funcionando sin ningún tipo de modificación.
- Definir como el acceso RDSI coexiste con especificaciones Bluetooth que posibilitan el acceso a RDSI de una manera o de otra.
- Mostrar como RDSI sobre tecnología Bluetooth puede coexistir con otras aplicaciones que ya emplean RDSI con anterioridad.

2.2.6.6 Perfil de telefonía sin cables (CTP)

Permite que los teléfonos móviles empleen enlaces Bluetooth. Se pensó para que dispositivos Bluetooth puedan conectarse a la línea de teléfono dentro de una casa y con la red de telefonía móvil cuando esta no esté disponible (fuera de la vivienda).

2.2.6.7 Perfil de conexión a red por dial-up o línea conmutada (DUN)

Proporciona un estándar para acceder a Internet y otros servicios dial-up sobre la tecnología Bluetooth. Se basa en SPP y el caso típico de uso es el de un portátil accediendo a Internet empleando la línea de un teléfono móvil.

2.2.6.8 Perfil de Fax (FAX)

Ofrece un interfaz entre un teléfono móvil y un ordenador con software instalado de fax. No cubre llamadas de voz ni datos.

2.2.6.9 Perfil de transferencia de ficheros (FTP)

Define como intercambiar ficheros y carpetas entre dos dispositivos. Una vez que el dispositivo cliente conoce el archivo o la ubicación, se puede enviar un archivo del servidor al cliente o desde el cliente al servidor usando GOEP. Emplea OBEX en la capa de transporte y también permite el listado de directorios, escritura y borrado de los mismos.

2.2.6.10 Perfil de distribución general de audio y video (GAVDP)

Proporciona la base a los perfiles A2DP y VDP.

2.2.6.11 Perfil de intercambio genérico de objetos (GOEP)

Se emplea para transferir objetos de un dispositivo a otro. Sirve como base para otros perfiles Bluetooth y está basado en OBEX.

2.2.6.12 Perfil de reemplazo de cables (HCRP)

Define como establecer un driver de impresora empleando la tecnología Bluetooth. Su utilidad se ve reducida en dispositivos sencillos ya que no disponen de un repositorio específico de drivers.

2.2.6.13 Perfil de manos libres (HFP)

Comúnmente se usa para recibir llamadas desde un dispositivo móvil a un dispositivo de manos libres. El escenario más común donde se emplea este perfil es dentro de un coche.

2.2.6.14 Perfil de dispositivo de interfaz humana (HID)

Define los protocolos, los procedimientos y características que dan soporte a dispositivos como por ejemplo a ratones, teclados y joysticks. Este perfil emplea el bus USB y está diseñado para proporcionar un enlace de baja latencia manteniendo un bajo consumo.

2.2.6.15 Perfil de auriculares (HSP)

Describe como unos auriculares con tecnología Bluetooth deben comunicarse con un ordenador u otros dispositivos Bluetooth como puede ser un teléfono móvil. Es un perfil muy usado en la actualidad.

2.2.6.16 Perfil Intercom (ICP)

Define como teléfonos móviles configurados en una misma red pueden comunicarse directamente entre ellos sin emplear la red de telefonía pública. Esta función habilita, por ejemplo, la interconexión de teléfonos con una oficina. Este perfil también se conoce como perfil "*walkie-talkie*".

2.2.6.17 Perfil básico de envío de objetos (OPP)

Se utiliza para enviar objetos genéricos como vCards, citas o fotos. El emisor es siempre el que inicia la comunicación, cumpliendo por completo el modelo Push. Utiliza las APIs de OBEX para las operaciones de conexión, desconexión, envío, recepción y cancelación.

2.2.6.18 Perfil de redes de área personal (PAN)

Describe como uno o más dispositivos Bluetooth pueden formar una red ad-hoc y como el mismo mecanismo se puede usar para acceder a una red remota a través de un punto de acceso.

2.2.6.19 Perfil de acceso a agenda telefónica (PBAP)

Permite el envío de agendas telefónicas entre dispositivos. También se emplea, en el caso de recibir una llamada, para transmitir información sobre la llamada entrante a otro dispositivo Bluetooth.

2.2.6.20 Perfil de descubrimiento de servicios (SDAP)

Describe como una aplicación debería usar SDP para descubrir servicios en un dispositivo remoto. Implementa varios enfoques para la gestión de descubrimiento de dispositivos a través del mecanismo *Inquiry* y vía *Inquiry Scan* descubre los servicios usando SDP. Los casos de uso para SDAP pretenden abarcar la mayoría de los escenarios de descubrimiento de servicios asociados con todos los perfiles y dispositivos. SDAP requiere que cualquier aplicación sea capaz de averiguar qué servicios están disponibles en cualquier dispositivo Bluetooth al que se conecte. El perfil se encarga tanto de la búsqueda de servicios específicos, como de la búsqueda de servicios más generales.

2.2.6.21 Perfil de puerto serie (SPP)

Define como configurar los puertos series virtuales y conectar así dos dispositivos Bluetooth. Se definen dos roles:

- *El dispositivo A:* es el dispositivo que toma la iniciativa para establecer la conexión con otro dispositivo (iniciador).
- *El dispositivo B:* espera a que otro dispositivo se conecte a él (aceptador).

Está basado en RFCOMM que es la adaptación Bluetooth GSM TS 07.10 que proporciona un protocolo de transporte para la simulación de un puerto serie.

En el apartado 2.2.7 se abordará más en profundidad dicho perfil ya que es el empleado en el marco del proyecto. En concreto, se hablará de su arquitectura y los diferentes protocolos que se emplean.

2.2.6.22 Perfil de Sincronización (SYNC)

Este perfil se usa en conjunto con GOEP para habilitar la sincronización de información de calendarios y direcciones entre dispositivos Bluetooth. Su origen es parte de la especificación de infrarrojo.

2.2.6.23 Perfil de distribución de video (VDP)

Habilita el transporte de un stream de video. Se emplea para distribuir un vídeo grabado desde cualquier dispositivo.

2.2.7 Arquitectura de protocolos en el perfil SPP

El perfil SPP [SPP 01] define los protocolos y procedimientos que deben usar dispositivos Bluetooth para simular un cable serie RS-232. El escenario típico hace referencia a aplicaciones heredadas que emplean el Bluetooth como un sustituto del cable, a través de una abstracción del puerto serie virtual.

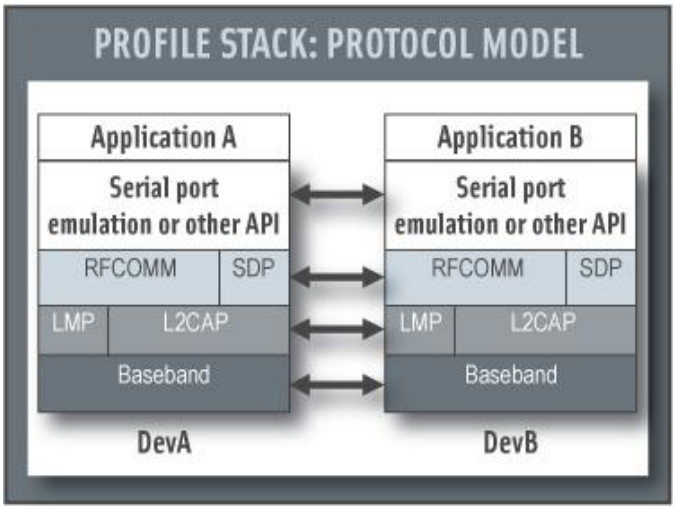


Figura 7: Protocolos y entidades usadas en el perfil SPP [Bluetooth Perfiles 12].

2.2.7.1 Banda Base

Cada dispositivo Bluetooth debe ser localizado de manera única por una dirección de dispositivo Bluetooth de 48 bits (BD_ADDR). Esta dirección se obtiene de la Autoridad de Registro IEEE y se compone de tres campos:

- *Campo LAP (24 bits)*: es la parte de la dirección con menos peso.
- *Campo UAP (8 bits)*: es la parte superior de la dirección.
- *Campo NAP (16 bits)*: es la parte no significativa de la dirección.

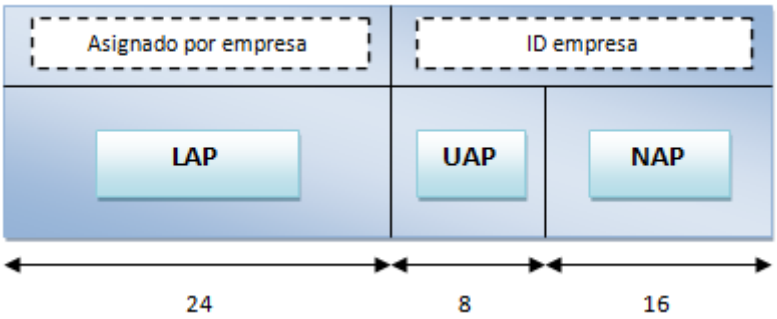


Figura 8: Formato de BD_ADDR.

En el sistema Bluetooth todas las transmisiones sobre el canal físico comienzan con un código de acceso. El código de acceso también indica al receptor la llegada de un paquete. Hay 3 códigos de acceso diferentes:

- *Código de acceso del dispositivo (DAC)*: se usa durante los subestados *page, page scan* y *page response.*
- *Código de acceso del canal (CAC)*: se emplea en el estado de conexión, es el comienzo del intercambio de paquetes en el canal físico de la *piconet*.
- *Código de acceso de búsqueda (IAC)*: se usa en el subestado *inquiry*.

La capa inferior de la arquitectura Bluetooth es el canal físico. Todos los canales físicos se caracterizan por la combinación de una secuencia de salto de frecuencia pseudo-aleatoria, el tiempo de slot específico de las transmisiones y el código de acceso y la codificación de la cabecera del paquete. Un dispositivo Bluetooth solo puede usar un canal físico al mismo tiempo. Para soportar múltiples operaciones a la vez, el dispositivo emplea multiplexación por división en el tiempo entre los canales. Se definen los siguientes canales físicos:

- *Canal físico básico de piconet*: se usa para la comunicación entre dispositivos conectados y está asociado con una *piconet* específica.
- *Canal físico adaptado de piconet*: también se usa para la comunicación entre dispositivos conectados y está asociado con una *piconet* específica. Las diferencias con el canal anterior son: primero, el mismo mecanismo de canal hace que la frecuencia del esclavo sea la misma que la frecuencia anterior de transmisión del maestro.

Segundo, este canal emplea menos frecuencias totales que el canal físico básico de *piconet*.

- *Canal físico de rastreo por llamada*: permite a los dispositivos conectables, capaces de aceptar conexiones, escuchar peticiones de comunicación.
- *Canal de rastreos por inspección*: se utiliza para descubrir dispositivos externos enviando peticiones en el rango de frecuencias y escuchando posibles respuestas.

Un enlace físico representa una conexión en banda pase entre dispositivos. Un enlace físico siempre está asociado con un canal físico. Los enlace físicos tienen propiedades en común que aplican a los transportes lógicos en la capa física. Estas son:

- *Control de potencia.*
- *Supervisión del enlace*
- *Encriptación*
- *Canal de cambio de tasa de datos basado en la calidad.*
- *Control de paquete multi-slot.*

Entre un maestro y uno o varios esclavos se pueden establecer 5 tipos diferentes de transportes lógicos:

- *Transporte lógico síncrono orientado a la conexión (SCO).*
- *Transporte lógico extendido síncrono orientado a la conexión (eSCO).*
- *Transporte lógico asíncrono orientado a la conexión (ACL).*
- *Transporte lógico broadcast de esclavo activo (ASB).*
- *Transporte lógico broadcast de esclavo pasivo (PSB).*

Los transportes lógicos síncronos, son transportes lógicos punto a punto entre un maestro y un único esclavo de la *piconet*. Típicamente soportan información por un tiempo limitado como voz o datos síncronos generales.

El transporte lógico ACL es también un transporte lógico entre un maestro y un único esclavo de la *piconet*. El ASB es usado por el maestro para comunicarse con esclavos activos y con el PSB el maestro se comunica con los esclavos pasivos.

En la banda base se definen 5 enlaces lógicos:

- *Enlace lógico de control del enlace (LC)*: este enlace lógico transporta información de control del enlace de nivel bajo como ARQ, control de flujo y caracterización de la

carga útil. El enlace lógico LC se transporta en cada paquete excepto en el paquete ID, ya que no contiene cabecera.

- *Enlace lógico de control ACL (ACL-C)*: intercambia información de control entre los gestores del enlace del maestro y del esclavo (s).
- *Enlace lógico asíncrono/isócrono de usuario (ACL-U)*: transporta datos de usuario L2CAP asíncronos e isócronos. Estos mensajes puedes transmitirse en uno o más paquetes banda base.
- *Enlace lógico de datos síncronos de usuario (SCO-S)*: este enlace lógico transporta datos síncronos de usuario sobre un transporte lógico SCO.
- *Enlace lógico extendido de datos síncronos de usuario (eSCO-S)*: transporta datos síncronos de usuario sobre un transporte lógico eSCO.

Hay dos tipos de paquetes con formato general: el formato de tasa básica y el formato de velocidad de datos mejorada.

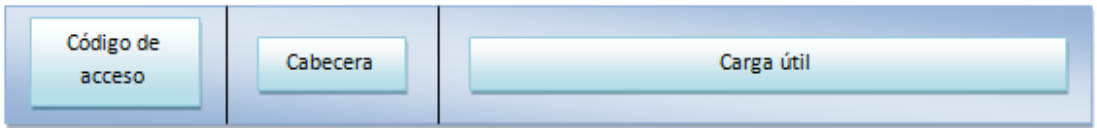


Figura 9: Formato del paquete de tasa básica.

Figura 10: Formato del paquete de velocidad de datos mejorada.

En la siguiente figura se muestran los tres estados principales del controlador de enlace: *Standby, Connection* y *Park*. Además, hay siete subestados*: page, page scan, inquiry, inquiry scan, master response, slave response* e *inquiry response*. Los subestados son estados internos que se usan para establecer conexiones y descubrir dispositivos que estén disponibles.

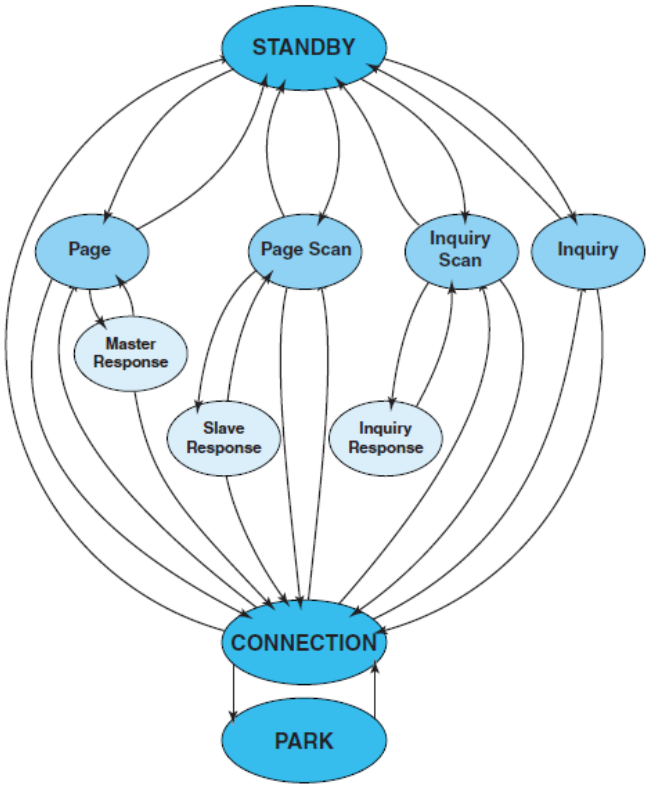


Figura 11: Diagrama de estados del controlador del enlace [Specification 2.1 + EDR 07].

El estado *Standby* es el estado por defecto en el dispositivo. En este estado, el dispositivo puede entrar en un modo de ahorro de energía. El controlador puede abandonar este estado para escanear dispositivos o consultar mensajes.

En el subestado *page scan*, el dispositivo puede ser configurado para usar tanto el procedimiento de escaneo estándar como el entrelazado. El subestado *page* lo emplea el maestro (origen) para activar y conectarse a un esclavo (destino) que se encuentra en el subestado *page scan*. Cuando un mensaje *page* es recibido satisfactoriamente por el esclavo, hay una sincronización entre el maestro y el esclavo. Ambos, entran en un subestado *response* para intercambiar información esencial para continuar con la configuración de la conexión. Con el fin de descubrir otros dispositivos, un dispositivo puede entrar en el subestado *inquiry*. En este subestado, se puede enviar repetidamente el mensaje *inquiry* en diferentes frecuencias de salto.

El subestado *inquiry scan* es muy similar al subestado *page scan*. Sin embargo, en lugar de escanear el código de acceso de los dispositivos, el receptor escanea el código de acceso de solicitud. El subestado *inquiry* se emplea para descubrir nuevos dispositivos. Este subestado es muy similar al del subestado *page*. En el estado *connection*, la conexión ya se ha establecido y

los paquetes pueden enviarse bidireccionalmente. Cuando un esclavo no necesita participar en el canal de la *piconet*, pero todavía necesita permanecer sincronizado con el canal, puede entrar en el estado *park*. Este estado conlleva muy poca actividad en el esclavo.

2.2.7.2 LMP

El protocolo de gestión de enlace (LMP) se usa para controlar y negociar todos los aspectos de la operación de conexión Bluetooth entre dos dispositivos. Incluye [Specification 2.1 + EDR 07] la configuración y control de los enlaces lógicos y el control de los transportes lógicos y para controlar los enlaces físicos. El protocolo LMP se emplea para la comunicación entre la capa de gestión (LM) de dos dispositivos que están conectados a través del transporte lógico ACL.

Todos los mensajes LMP se aplicarán únicamente al enlace físico y a los enlaces lógicos asociados y a los transportes lógicos entre receptores y emisores. El protocolo se compone de una serie de mensajes los cuáles se transfieren sobre el enlace lógico ACL-C en el transporte lógico ACL por defecto entre dos dispositivos. Dichos mensajes LMP se intercambian sobre el enlace lógico ACL-C, que se distingue del enlace ACL-U (el cual lleva datos de usuario y L2CAP) por el identificador de enlace lógico (LLID) que aparece en la cabecera de la carga útil de los paquetes de longitud variable.

Las capacidades de detección y corrección de errores del transporte lógico ACL de la banda base, generalmente, son suficientes para las necesidades de LMP. El enlace lógico ACL-C no garantiza un tiempo de entrega ni paquetes de confirmación Ack. Los procedimientos LMP tienen esto en cuenta cuando el estado de sincronización cambia en ambos dispositivos.

A cada PDU se le asigna un código de operación de 7 ó 15 bits para identificar de forma única los diferentes tipos de PDUs. Los primeros 7 bits del código de operación y un identificador de transacción se encuentran en el primer byte del cuerpo donde se aloja la carga de datos. Si la PDU contiene uno o más parámetros, estos se colocan en la carga que va inmediatamente después de del código de operación.

Figura 12: PDU LMP con 7 bit de código de operación (arriba) y PDU LMP con 15 bit de código de operación (abajo).

El protocolo LMP funciona a través de transacciones. Una transacción es un conjunto conectado de intercambios de mensajes que logran un propósito particular. Todas las PDUs que forman parte de la misma transacción tienen que tener el mismo valor de ID de transacción. Este campo debe ser 0 si la PDU forma parte de una transacción que ha sido iniciado por el maestro y 1 si ha sido el esclavo quien lo ha iniciado.

El tiempo que se tarda desde que se recibe un paquete de banda base que lleva una PDU LMP hasta que se envía un paquete banda base con la respuesta válida LMP debe ser inferior que el Tiempo de Espera de Respuesta LMP que se corresponde con 30 segundos.

Si LM recibe una PDU con un código de operación no conocido hay que responder con un *LMP_not_accepted* o un *LMP_not_accepted_ext* con el código de error *unknown LMP PDU.* Si LM recibe una PDU con parámetros inválidos, se enviará un *LMP_not_accepted* o un *LMP_not_accepted_ext* con el código de error *invalid LMP parameters*. Si se excede el tiempo máximo de respuesta o si se detecta pérdida del enlace la parte que espera una respuesta debe concluir que el procedimiento ha terminado sin éxito.

En ocasiones, pueden ocurrir situaciones de colisión cuando distintos LMs inician el mismo procedimiento y no pueden completarlo. En esta situación, el maestro debe descartar el procedimiento iniciado por el esclavo enviando un *LMP_not_accepted* o un *LMP_not_accepted_ext* con el código de error *LMP error transaction collision*. El procedimiento iniciado por el maestro se completará. Una situación de colisión también puede ocurrir cuando ambos LMs inician diferentes procedimientos pero ninguno de los dos puede completarlos.

Las características en LMP se representan mediante una máscara de bits. Cada bit representa una característica, si se pone a 1 esta se activa. La única excepción se da con el *flow control lag* que está codificado con 3 bits.

LMP también se encarga de tareas de seguridad, como son la autenticación y la encriptación. En el proceso de autenticación, el verificador envía una PDU *LMP_au_rand* que contiene un número aleatorio (el reto) a la entidad demandante. Este calcula una respuesta, que es una función que incluye los parámetros: reto, BD_ADDR del demandante y una clave secreta. Se devuelve la respuesta al verificador, el cual comprueba si la respuesta es correcta o no.

En lo que se refiere a la encriptación, al menos se puede realizar una autenticación con cifrado. Para que el maestro use los mismos parámetros de encriptación en todos los esclavos de la *piconet* debe usar una clave temporal, K_{master}. El maestro debe establecer esta clave como la clave del enlace actual para todos los esclavos de la *piconet* antes de que comience la encriptación.

En LMP existen 3 modos de operación:

- *Modo de espera*: el transporte lógico ACL de una conexión entre dos dispositivos Bluetooth puede tener lugar en modo de espera durante un tiempo determinado de espera. Ambas PDUs, *LMP_hold* y *LMP_hold_req*, contienen un parámetro "instante de espera" que especifica en qué momento el modo de espera es efectivo. Este parámetro es especificado como un valor de reloj Bluetooth del reloj del maestro, que está disponible en ambos dispositivos.
- *Modo de estado aparcado*: si un esclavo no necesita participar en un canal, pero debería permanecer sincronizado con el maestro, debe emplear este modo. Todas las PDUs que se envían desde el maestro a los esclavos aparcados lo hacen por el enlace lógico PSB-C (enlace LMP de transporte lógico broadcast de esclavo aparcado). Estas PDUs, *LMP_set_broadcast_scan_window, LMP_modify_beacon, LMP_unpark_BD_addr_req, LMP_unpark_PM_addr_req*, son solo PDUs que deben enviarse a un esclavo es estado aparcado y las únicas que son broadcast.
- *Modo sniffer*: para entrar en este modo, el maestro y el esclavo negocian un intervalo de sniff, T_{sniff} y un offset de sniff, D_{sniff}, que especifica el tiempo de los slots del sniff. Cuando el transporte lógico ACL está en modo sniff el maestro solo debe comenzar una transmisión en los slots del sniff.

Cuando una conexión entre dos dispositivos se establece por primera vez, la conexión consiste en enlaces lógicos ACL: ACL-C, para mensajes LMP y ACL-U para datos L2CAP. También hay que añadir uno o más transportes lógicos síncronos (SCO o eSCO).

También hay que destacar que LMP dispone de PDUs que soportan diferentes modos de pruebas que se usan para certificación y pruebas de conformidad en la interfaz radio Bluetooth y en la banda base.

2.2.7.3 L2CAP

Es un protocolo de control y adaptación del enlace lógico [Specification 2.1 + EDR 07]. Proporciona servicios de datos orientados a la conexión y no orientados a la conexión a los protocolos de capas superiores con capacidades de multiplexación, segmentación y reensamblado. L2CAP permite a los protocolos de capas superiores y a las aplicaciones transmitir y recibir paquetes de datos (SDU) hasta una longitud de 64 Kbytes. L2CAP también permite control de flujo por canal y retransmisión. L2CAP dispone de canales lógicos, llamados canales L2CAP, los cuales son mapeados a enlaces lógicos L2CAP soportados por un transporte lógico ACL.

El Administrador de Canal proporciona la funcionalidad del plano de control y es el responsable de toda la señalización interna, señalización L2CAP punto a punto y señalización con respecto a las capas inferiores y superiores. El Administrador de Recursos es el responsable de proporcionar un servicio de Frame Relay al Administrador de Canal.

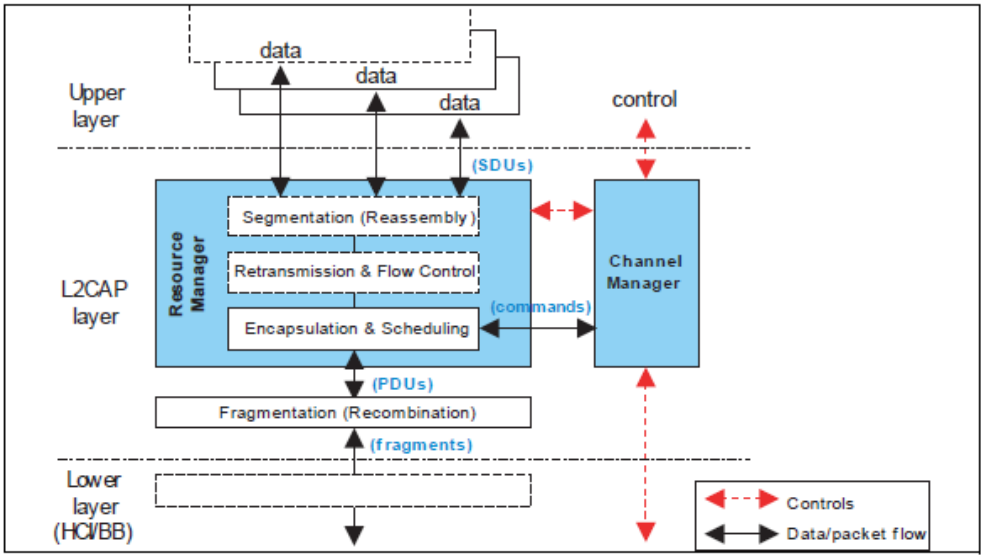


Figura 13: Arquitectura de bloques L2CAP [Specification 2.1 + EDR 07].

Las características principales de L2CAP son:

- *Multiplexaición protocolo/canal*: L2CAP soporta multiplexación porque el protocolo de banda base no soporta ningún tipo de campo para la identificación del protocolo de capa superior. Durante la configuración del canal, la capacidad de multiplexación se utiliza para encaminar la solicitud de conexión al protocolo correcto de la capa superior. Durante la transferencia de datos, se necesita la multiplexación del canal lógico para conseguir distinguir entre múltiples entidades de la capa superior.
- *Segmentación y reensamblado*: con el servicio Frame Relay ofrecido por el Administrador de Recursos, la longitud de las tramas de transporte son controladas por las aplicaciones individuales que corren sobre L2CAP. Muchas aplicaciones son atendidas mejor si L2CAP tiene control sobre la longitud de la PDU. Las ventajas son:
    1. La segmentación permitirá el intervalo de unidades de datos de aplicación para satisfacer los requerimientos de latencia.
    2. La administración de la memoria y el buffer es más fácil cuando L2CAP controla el tamaño del paquete.
    3. La conexión de error por retransmisión puede ser más eficiente.
    4. La cantidad de datos que se desechan cuando una PDU L2CAP está corrupta o se ha perdido puede ser más pequeña que unidad de datos de aplicación.
- *Control de flujo por cada canal L2CAP*: cuando varios streams de datos corren sobre el mismo enlace lógico L2CAP, usando canales L2CAP separados, cada canal puede requerir flujo de control individual. L2CAP también puede proporcionar servicios de

control de flujo para perfiles y aplicaciones que lo necesiten y que quieran evitar implementarlo ellas mismas.

- *Control de error y retransmisiones*: algunas aplicaciones necesitan una tasa de error residual mucho más pequeña que la entregada por la banda base. L2CAP incluye, de manera opcional, comprobación de errores y retransmisiones de cada PDU L2CAP. La comprobación de errores en L2CAP protege contra errores causados por la banda base al aceptar cabeceras de paquetes erróneas y causados, también, por los fallos en el CRC de los paquetes. El modo de retransmisión también protege contra la perdida de paquetes.
- *Fragmentación y recombinación*: las capas inferiores tienen bajas capacidades de transmisión y pueden requerir fragmentación en diferentes tamaños. Además estas capas pueden aun fragmentar más y recombinar PDUs L2CAP para crear fragmentos que se adapten a cada nivel de la arquitectura. Durante la transmisión de una PDU L2CAP pueden ocurrir diferentes ocasiones donde tengan lugar las operaciones de segmentación y reensamblado en ambos dispositivos.
- Calidad de servicio (QoS): el proceso de establecimiento de la conexión permite el intercambio de información manteniendo la QoS esperada entre dos dispositivos Bluetooth.

Como hemos mencionado anteriormente, L2CAP se basa en el concepto de los canales. Un identificador de canal (CID) es el nombre local que representa al endpoint del canal lógico del dispositivo. El identificador nulo (0x0000) es un identificador legal y nunca se debe usar como un endpoint de destino. Los identificadores en el rango 0x0001-0x003F están reservados para funciones específicas L2CAP. Entre los CIDs 0x0040 y 0xFFFF se puede elegir el que se quiera, siempre que no estén activos de manera simultánea dos canales L2CAP con el mismo CID. La asignación del CID es relativa a un dispositivo en particular y un dispositivo puede asignar CIDs de manera independiente con respecto a otros dispositivos.

L2CAP puede operar en 3 modos de operación diferentes. Los modos son:

- *Modo básico L2CAP*: es el modo por defecto.
- *Modo de control de flujo*: no se producen retransmisiones, la pérdida de PDUs se detectan y se puede informar sobre su pérdida.
- *Modo de retransmisión*: se usa un temporizador para asegurar que todas las PDUs se han entregado al destino, retransmitiendo las PDUs cuando sea necesario. Se emplea

un mecanismo de repetición vuelta-atrás-n para simplificar el protocolo y limitar las necesidades del buffer.

En el modo básico L2CAP, la PDU en un canal orientado a la conexión también se la denomina "*B-frame*".

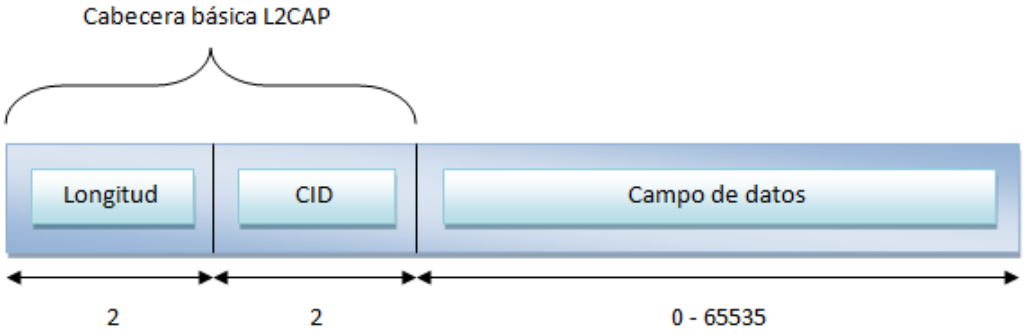


Figura 14: Estructura de la PDU "*B-frame*".

Como se observa en la Figura 14, los campos que forman la PDU *"B-frame"* son:

- *Longitud (2 octetos)*: indica el tamaño del campo de datos en octetos, excluyendo la cabecera L2CAP.
- *CID (2 octetos)*: identifica el endpoint del canal destino del paquete.
- *Campo de datos (0 - 65535 octetos)*: contiene los datos recibidos del protocolo de la capa superior (paquete entrante) o los entrega a la capa superior (paquete saliente).

En el modo básico L2CAP, la PDU en un canal de datos no orientado a la conexión también se le denomina *"G-frame"*.

Los campos que lo forman son: longitud, CID (0x0002), Multiplexor protocolo/servicio (PSM) y el campo de datos.

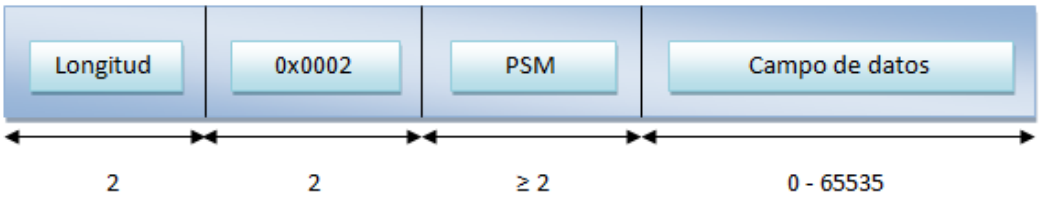


Figura 15: Estructura de la PDU "*G-frame*".

En los modos de control de flujo y retransmisión con un canal orientado a la conexión existen dos tipos de tramas: las tramas de información (*I-frames*) y las tramas de supervisión (*S-frames*).

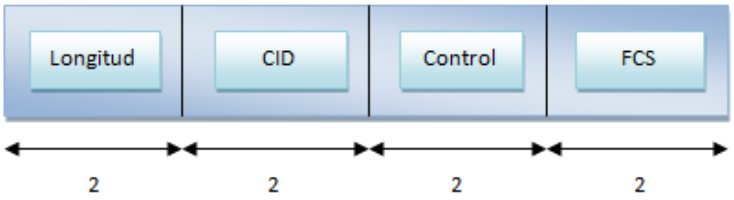


Figura 16: Estructura de la PDU "*S-frame*".

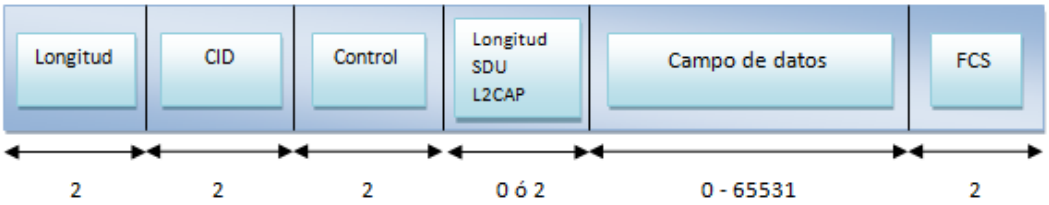


Figura 17: Estructura de la PDU "*I-frame*".

Los campos que forman las tramas son:

- *Longitud (2 octetos).*
- *CID (2 octetos).*
- *Campo de control (2 octetos)*: identifica el tipo de trama.
    - *I-frame*: se usan para transferir información entre entidades L2CAP. Cada *I-frame* tiene un *TxSeq* (número de secuencia enviado), *ReqSeq* (número de secuencia recibida), un bit de retransmisión (*bit R*) y el campo *SAR* que se emplea para el control de la segmentación y el reensamblado.
    - *S-frame*: se emplean para la confirmación de tramas I y solicitar, cuando sea necesario, su retransmisión. Cada *S-frame* tiene un *ReqSeq* y un bit *R*. Los tipos de *S-frames* definidos son *RR* (Receptor preparado) y *REJ* (descartar).
- *Campo de longitud de la SDU L2CAP (0 ó 2 octetos)*: cuando una SDU ocupa más de una trama *I-frame*, la primera trama I de la secuencia se identifica con el campo *SAR=01= "Comienzo de la SDU L2CAP"*. El campo de longitud de la SDU L2CAP especifica el número total de octetos de la SDU. Este campo solo se usa en las tramas con SAR=01, en el resto de trama I no está presente.
- *Campo de datos (0 – 65531 octetos).*

- *FCS (Secuencia de verificación de trama, 2 octetos):* se construye usando el polinomio generador g (D) = $D^{16} + D^{15} + D^2 + 1$.

Una PDU recibida, de las mencionadas anteriormente, se descartará cuando, al menos, ocurra una de las siguientes condiciones.

1. Contenga un CID desconocido.
2. Contenga un error de FCS.
3. Contenga una longitud mayor al tamaño máximo del campo de datos de la PDU (MPS).
4. Que una *I-frame* tenga menos de 8 octetos.
5. Que una *I-frame* con SAR=01 tenga menos de 10 octetos.
6. Si en una trama I aparecen bits SAR y esta trama no es una trama de "Comienzo de la SDU L2CAP".
7. Si en una trama S el campo de longitud es distinto de 2 bytes.

La información de señalización se envía mediante *C-frames* (tramas de control) por el canal de señalización con el CID=0x0001. Este canal de señalización está disponible en cuánto se configura el transporte lógico ACL y se habilita el tráfico L2CAP en el enlace lógico L2CAP. En una *C-frame* se pueden enviar múltiples comandos.

Figura 18: Trama de control (*C-frame*).

Como se observa en la Figura 19, un comando de señalización está formado por los siguientes campos:

- *Código (1 octeto):* identifica el tipo de comando. Cuando se recibe un paquete con un código desconocido se envía como respuesta un paquete "*Command Reject*".
- *Identificador (1 octeto):* empareja las respuestas con las solicitudes.
- *Longitud (2 octetos)*: indica el tamaño en octetos exclusivamente del campo de datos de comandos.
- *Datos (0 o más octetos)*: el código determina el formato del campo de datos.

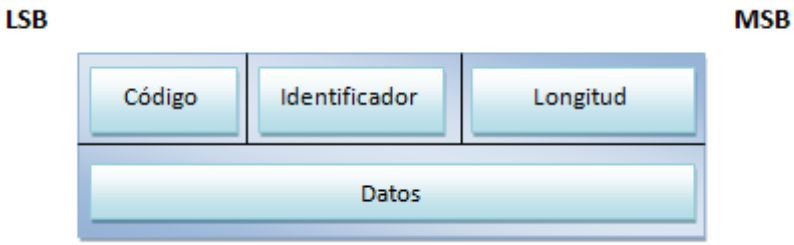


Figura 19: Formato de un comando.

Los comandos empleados son:

- *Comando Reject (Código 0x01)*: se envía como respuesta a un comando que tiene un código desconocido, o bien cuando al enviar la respuesta correspondiente es inapropiada. Tiene un campo denominado "*Razón*" que se emplea para describir la razón de por qué el paquete ha sido descartado (*0x0000* comando no entendible, *0x0001* señalización MTU excedida, *0x0002* CID de solicitud inválido).
- *Comando Connection Request (Código 0x02)*: este comando se envía para crear un canal L2CAP entre dos dispositivos. El canal L2CAP debe estar establecido antes de que comience su configuración. Cabe distinguir los campos PSM y SCID. PSM es un campo de al menos 2 octetos en el que sus valores se separan en dos rangos; el primero es asignado por el Bluetooth SIG y sirve para indicar los protocolos; el segundo es asignado dinámicamente y se usa en conjunto con SDP. El campo SCID (CID origen) tiene 2 octetos de longitud y representa el extremo del canal en el dispositivo que recibe la solicitud.
- *Comando Connection Response (Código 0x03)*: cuando un dispositivo recibe un paquete *Connection Request*, debe responder con dicho comando. Los campos que forman el campo de datos son:
    - *DCID (Identificador del canal de destino, 2 octetos):* contiene el extremo del canal desde el que el dispositivo envía el comando.
    - *SCID (2 octetos).*
    - *Resultado (2 octetos):* indica el resultado de la solicitud de conexión. Los valores puede ser: *0x0000* (conexión exitosa), *0x0001* (conexión pendiente), *0x0002* (PSM no soportada), *0x0003* (bloqueo por seguridad) y *0x0004* (recursos no disponibles).
    - *Estado (2 octetos)*: indica el estado de la conexión.
- *Comando Configuration Request (Código 0x04):* estos paquetes se envían para establecer un enlace lógico inicial de transmisión entre dos entidades L2CAP y también

para renegociar este convenio cuando sea necesario. Durante la sesión de renegociación, todo el tráfico de datos en el canal debe ser suspendido mientras se espera el resultado de la negociación. Los campos de datos son:

  - *DCID (2 octetos).*
  - *Flags (2 octetos):* solo se define un tipo de flag, el flag de continuación *(C)*, que indica que la respuesta debe esperar a recibir varios paquetes de solicitudes.
  - *Opciones de configuración*: es una lista de los parámetros con los valores que deben ser negociados.

- *Comando Configuration Response (Código 0x05)*: los paquetes se envían en respuesta a los paquetes *Configuration Request* excepto cuando ocurre un error, que entonces se envía un *Reject Response*. Los campos de datos son:
  - *SCID (2 octetos).*
  - *Flags (2 octetos).*
  - *Resultado (2 octetos):* indica si la solicitud ha sido o no aceptada. El resultado *0x0000* indica éxito, *0x0001* indica parámetros no aceptados, *0x0002* significa solicitud descartada *y 0x0003* indica que las opciones son desconocidas.
- *Comando Disconnection Request (Código 0x06):* terminar un canal L2CAP requiere que se envíe una solicitud de desconexión y una Ack como respuesta. Una vez que se emite un comando *Disconnection Request* todos los datos entrantes y salientes en tránsito en este canal L2CAP serán descartados. Los campos de datos son:
  - *DCID (2 octetos).*
  - *SCID (2 octetos).*
- *Comando Disconnection Response (Código 0x07):* se envía como respuesta a una solicitud de desconexión válida. Los campos de datos son:
  - *DCID (2 octetos).*
  - *SCID (2 octetos).*
- *Comando Echo Request (Código 0x08):* se emplea para solicitar una respuesta desde una entidad L2CAP remota. Puede emplearse para testear el enlace o para enviar información específica del fabricante usando el campo de datos, que en este caso es opcional.
- *Comando Echo Response (Código 0x09):* se envía una vez recibida una solicitud *Echo* válida.
- *Comando Information Request (Código 0x0A):* se emplea para solicitar información específica de implementación desde una entidad L2CAP remota. El campo de datos

está formado por el campo *InfoType* de 2 octetos, el cuál define el tipo de información que se solicita (*0x0001*: MTU sin conexión y *0x0002*: funciones ampliadas soportadas).

- *Comando Information Response (Código 0x0B):* una vez recibida una solicitud válida de información se responde con un paquete *Information Response*. Los campos de datos son:
    - *InfoType (2 octetos).*
    - *Resultado (2 octetos)*: puede tomar el valor *0x0000* si la solicitud ha sido exitosa ó *0x0001* si no está soportada.
    - *Datos (0 ó más octetos)*: el contenido de este campo depende del campo *infoType*. Para *InfoType=0x0001* el campo de datos contiene 2 octetos de la MTU sin conexión aceptada por la entidad remota. Para el valor *0x0002*, el campo de datos contiene 4 octetos de la máscara de características extendidas. Esta máscara de bits es la forma de representar las características en el campo de datos del comando *Information Response*.

2.2.7.4 RFCOMM

El protocolo *Radio Frequency Communication* (RFCOMM) proporciona la simulación de los 9 circuitos de los puertos serie RS-232 (EIATIA-232-E) sobre el protocolo L2CAP. Este protocolo de transporte se basa en el estándar ETSI TS 07.10 y puede llegar a soportar hasta 60 conexiones simultáneas entre dos dispositivos Bluetooth [RFCOMM 03]. El número de conexiones, que se pueden usar a la vez, depende de la implementación específica de cada dispositivo Bluetooth.

Básicamente existen dos tipos de dispositivos en RFCOMM. Los dispositivos de Tipo 1, se corresponden con dispositivo finales como impresoras y ordenadores. Los de Tipo 2, son aquellos que forman parte del segmento de comunicación, por ejemplo los módems. Aunque en RFCOMM no se distinguen de manera explícita los dos tipos diferentes de dispositivos, a nivel de protocolo si tienen un efecto diferente.

La simulación de los puertos serie incluye la transferencia del estado de los circuitos que no se encargan de la transmisión de datos. Como se puede ver en la Tabla 2, los 9 circuitos RS-232 simulados en RFCOMM son:

PIN	NOMBRE DEL CIRCUITO
102	Señal Común
103	Transmitir datos (TD)
104	Recibir datos (RD)
105	Solicitud de envío (RTS)
106	Libre para envío (CTS)
107	Equipo de datos listo (DSR)
108	Terminal de datos listo (DTR)
109	Detección de portadora (DCD)
125	Indicador de llamada (RI)

Tabla 2: circuitos simulados RS-232 en RFCOMM

Dos dispositivos Bluetooth que emplean RFCOMM en su comunicación, pueden abrir múltiples puertos serie virtuales. RFCOMM soporta hasta 60 puertos abiertos. Sin embargo, el número de puertos depende del dispositivo Bluetooth que se esté utilizando. El identificador DLCI se utiliza para identificar una conexión establecida entre una aplicación servidora y un cliente. El DLCI se representa con 6 bits, aunque el rango que se utiliza es del 2…61; ya que el *DLCI 0* es el canal dedicado de control, el *DLCI 1* tampoco se emplea (Canales del servidor) y los *DLCIs 62 y 63* están reservados. El DLCI es único para cada sesión RFCOMM establecida entre dos dispositivos Bluetooth.

RFCOMM también es capaz de tener abiertas varias sesiones multiplexadas.

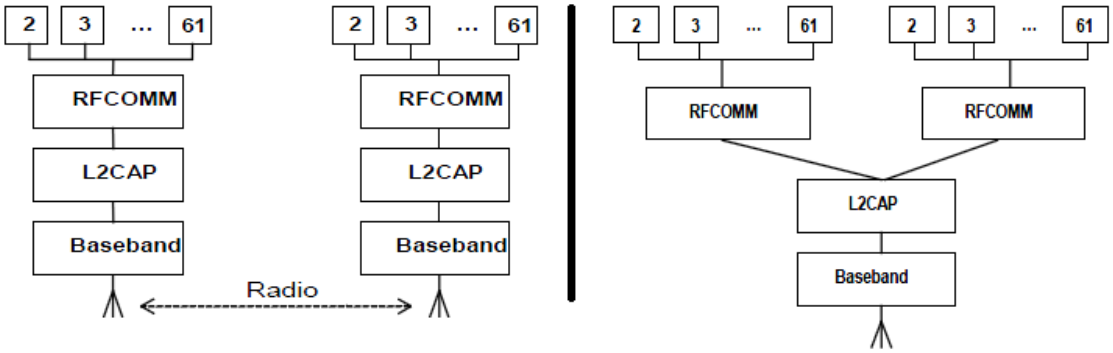


Figura 20: En RFCOMM, múltiples puertos serie simulados (izq.) y puertos serie procedentes de dos dispositivos Bluetooth diferentes (drcha.) [RFCOMM 03].

RFCOMM soporta diferentes tipos de tramas:

- *SABM:* comando para activar el modo asíncrono balanceado.
- *UA*: respuesta ACK sin numerar.
- *DM*: respuesta al modo de desconexión.

- *DISC*: comando de desconexión.
- *UIH*: comando y respuesta de información sin numerar con comprobación de cabecera.

TS 07.10 define un multiplexor que tiene un canal dedicado de control (*DLCI 0*). Este canal de control se emplea para transportar información entre dos multiplexores. En la Tabla 3 se muestran los comandos empleados:

COMANDOS	DESCRIPCIÓN
Test	Comando de test
Fcon	Comando para activar el control de flujo
Fcoff	Comando para desactivar el control de flujo
MSC	Comando del estado del modem
RPN	Comando de negociación del puerto remoto
RLS	Estado de la línea remota
PN	Negociación del parámetro DLC
NSC	Respuesta de comando no soportado

Tabla 3: Comandos RFCOMM.

- *RPN*: se usa antes de abrir un nuevo DLC y debería usarse siempre que la configuración del puerto cambie. Es un comando opcional en TS 07.10, pero obligatorio en RFCOMM.
- *RLS:* este comando es empleado para indicador el estado de la línea del puerto remoto. Es opcional en TS 07.10, pero obligatorio en RFCOMM.
- *PN*: es opcional en TS 07.10, pero obligatorio en RFCOMM. Se debe usar al menos antes de la creación del primer DLC en una sesión RFCOMM, y el iniciador tiene que intentar activar el control de flujo basado en el crédito.

La estructura básica de la trama es la misma que la trama de opciones básicas de 07.10 exceptuando los flags de apertura y cierre de la trama. Los campos que se intercambian entre las capas L2CAP y RFCOMM son: dirección, campo de control, indicador de longitud, información y FCS (Frame Check Sequence).

Flag	Address	Control	Length Indicator	Information	FCS	Flag
0111 1101	1 octet	1 octet	1 or 2 octets	Unspecified length but integral number of octets	1 octet	0111 1101

Figura 21: Estructura de la trama con opciones básicas [RFCOMM 03].

Los puertos cableados normalmente emplean control de flujo como RTS/CTS para controlar las comunicaciones. Por otro lado, el control de flujo entre RFCOMM y la capa inferior L2CAP depende de la interfaz de servicio soportada por la implementación. Además, RFCOMM tiene sus propios mecanismos de control de flujo:

- *Control de flujo L2CAP*: L2CAP se basa en el mecanismo de control de flujo proporcionado por la capa LM en la banda base.
- *Control de flujo de puerto serie cableado*: usa software de control de flujo usando caracteres como XON/XOFF y control de flujo usando los circuitos RTS/CTS o DTR/DSR.
- *Control de flujo GSM TS 07.10:* se diferencian dos mecanismo de control de flujo:
    1. El protocolo GSM TS 07.10 contiene comandos de control de flujo que operan en el flujo de datos agregados entre dos entidades RFCOMM.
    2. El comando del estado del modem es el mecanismo de control de flujo que opera en un DLCI individual.
- *Control de flujo serie de la entidad de simulación de puerto*: los dispositivos de Tipo 1, necesitan alguno drivers para proporcionar servicios de control de flujo como se especifica en la API donde se emulan. Una aplicación puede solicitar un mecanismo de control de flujo en particular como XON/XOFF o RTS/CTS y esperar a que el driver de puerto maneje el control de flujo. En los dispositivos de Tipo 2, el driver puede que necesite realizar control de flujo en la parte del path de comunicación que no es RFCOMM (puerto físico RS-232). Este control de flujo se especifica cuando los parámetros de control son enviados por la entidad RFCOMM.

2.2.7.5 SDP

El protocolo de descubrimiento de servicio (SDP) proporciona un medio para que las aplicaciones puedan descubrir aquellos servicios que están disponibles y determinar sus características.

SDP implica [Specification 2.1 + EDR 07] la comunicación entre un servidor SDP y un cliente SDP. El servidor mantiene una lista de registros de servicio que describe las características de los servicios asociados con el servidor. Cada registro de servicio contiene información sobre un único servicio. Un cliente puede recuperar información desde un registro de servicio que se aloja en el servidor SDP empleando una solicitud SDP.

Si el cliente decide usar un servicio, éste abre una conexión distinta para el proveedor de servicios con el fin de poder utilizar dicho servicio. SDP proporciona un mecanismo para el descubrimiento de servicios y sus atributos, pero no un mecanismo para utilizar dichos servicios.

Figura 22: Arquitectura del protocolo SDP.

Habrá como máximo un servidor SDP activo por dispositivo Bluetooth. Si un dispositivo Bluetooth actúa solo como cliente, no necesita un servidor SDP. Un único dispositivo Bluetooth puede funcionar tanto de servidor como de cliente SDP. Si múltiples aplicaciones de un dispositivo proporcionan servicios, un servidor SDP puede actuar en nombre de esos proveedores de servicio para gestionar las solicitudes SDP sobre los servicios que ellos proporcionan.

De manera similar, múltiples aplicaciones cliente pueden utilizar un cliente SDP para consultar servidores en nombre de las aplicaciones cliente.

El conjunto de servidores SDP que están disponibles para un cliente SDP cambiará de forma dinámica, basándose en la proximidad por radio frecuencia entre los servidores y el cliente. Cuando un servidor está disponible, se le notifica al cliente por medios diferentes a SDP de forma que el cliente puede usar SDP para solicitar servicios al servidor. Cuando un servidor deja de estar disponible por cualquier razón, no existe una notificación específica para indicarlo. Sin embargo, el cliente puede usar SDP para consultar al servidor y saber si está o no disponible.

Un servicio es una entidad que puede proporcionar información, realizar una acción, o controlar un recurso en nombre de otra entidad. Un servicio puede estar implementado como software, hardware o una combinación de ambos. Toda la información sobre un servicio se mantiene en un servidor SDP en un único registro de servicio. Este registro de servicio debe ser una lista de atributos de servicio.

Un registro de servicio es un número de 32 bits que identifica de manera única cada registro de servicio con respecto a un servidor SDP. El protocolo SDP no proporciona un mecanismo específico para notificar a los clientes cuando los registros de servicio son añadidos o eliminados del servidor SDP.

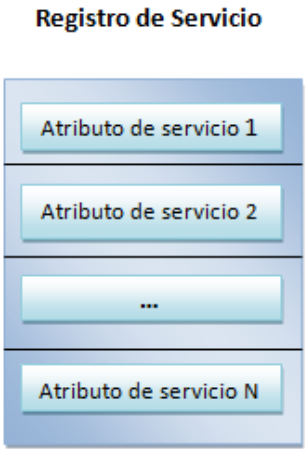


Figura 23: Registro de servicio en SDP.

Cada atributo de servicio (Figura 24) describe una única característica de un servicio. Cada atributo está compuesto por un ID de atributo y por un valor de atributo:

- Un ID de atributo es un entero de 16 bits que distingue cada atributo de servicio del resto de atributos que compone un registro de servicio. El atributo ID también identifica las semánticas del valor del atributo asociado. Cada ID se define de manera única solo dentro de cada clase de servicio y en SDP se representa como un "*Data Element*".
- El valor de atributo es un campo de longitud variable cuyo contenido está determinado por su atributo ID asociado y por la clase de servicio del registro de servicio en el cual está contenido el atributo. También se representa como un "*Data Element*".

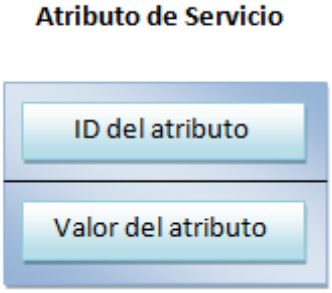


Figura 24: Atributo de servicio en SDP.

Cada servicio es una instancia de una clase de servicio. La definición de la clase de servicio proporciona las definiciones de todos los atributos que componen los registros de servicio que representan instancias de esa clase. Cada definición de atributo especifica el valor numérico del ID de atributo y el formato del valor de atributo. Un registro de servicio contiene atributos tanto específicos de una clase de servicio como atributos de carácter general común a todos los servicios.

A cada clase de servicio también se le asigna un identificador único, en concreto está contenido por el atributo "*ServiceClassIDList*" y es representado como un UUID.

La transacción de la búsqueda de servicio permite a un cliente recuperar un registro de servicio, manejado por registros de servicios particulares basados en los valores de atributos que están contenidos en ellos. Una vez que un cliente SDP tiene disponible un registro de servicio, puede solicitar los valores específicos de los atributos. Además, esta capacidad es proporcionada para buscar solo atributos cuyos valores son identificadores UUIDs.

Un UUID en un identificador único universal que garantiza ser único a lo largo del tiempo y el espacio. Los UUIDs pueden ser creados independientemente de una manera distribuida. No se requiere un registro central para asignar UUIDs. Un UUID es un valor de 128 bits. También existe un servicio de patrón de búsqueda, que consiste en una lista de UUIDs empleados para localizar los registros de servicio coincidentes.

Hay ocasiones en la que es deseable descubrir que tipos de servicios son descritos en un servidor SDP sin conocer nada sobre ellos previamente. Para ello existe un proceso llamado "*Navegación*". En SDP este mecanismo se basa en que todas las clases de servicio compartan un atributo. Este atributo se llama "*BrowseGroupList*" y su valor contiene una lista de UUIDs. Cada UUID representa a un grupo de navegación al cual se le asocia un servicio para encargarse del proceso de navegación.

SDP define un simple mecanismo para representar los datos con un ID de atributo, una distribución de IDs de atributo y un valor de atributo empleando un "*Data Element*". Un "*Data Element*" es un tipo de representación de datos. Está formado por dos campos:

- *El campo de cabecera* que a su vez se compone de:
    - *Tipo del descriptor*: se representa con 5 bits en la parte más significativa de la cabecera del Data Element.

- *Tamaño del descriptor*: se representa con un índice de 3 bits seguido por 0, 8, 16 o 32 bits. Se encuentra en la parte menos significativa de la cabecera del Data Element.

- *El campo de datos*: es una secuencia de bytes cuya longitud se especifica en el campo tamaño del descriptor y su contenido se especifica en el tamaño del descriptor.

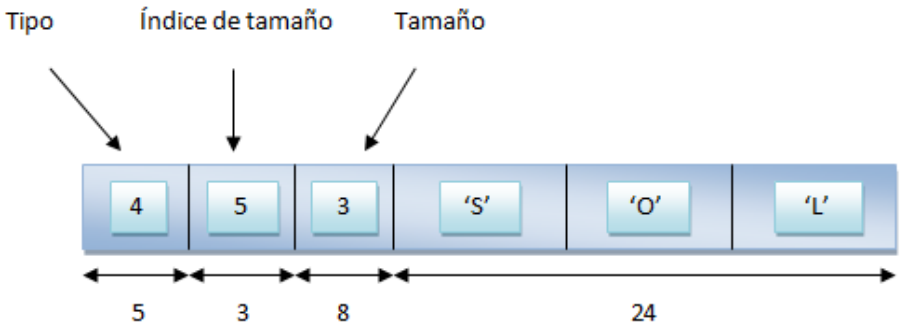


Figura 25: Representación de "Data Element" de 3 caracteres ASCII.

SDP es un protocolo sencillo con mínimos requerimientos de la capa subyacente de transporte. Puede funcionar sobre una capa de transporte de paquetes fiable. SDP emplea un modelo solicitud/respuesta donde cada transacción consiste en una PDU de solicitud y otra PDU de respuesta. En el caso de que SDP se use con el protocolo L2CAP, no se podrá enviar más de una solicitud SDP por conexión. En otras palabras, un cliente no puede emitir más de una solicitud a un servidor antes de recibir una respuesta a la solicitud anterior, una vez recibida la respuesta se podrá enviar otra solicitud.

Cada PDU SDP está formada por una cabecera y unos parámetros específicos. La cabecera contiene tres campos: ID de la PDU, ID de la transacción y longitud de los parámetros.

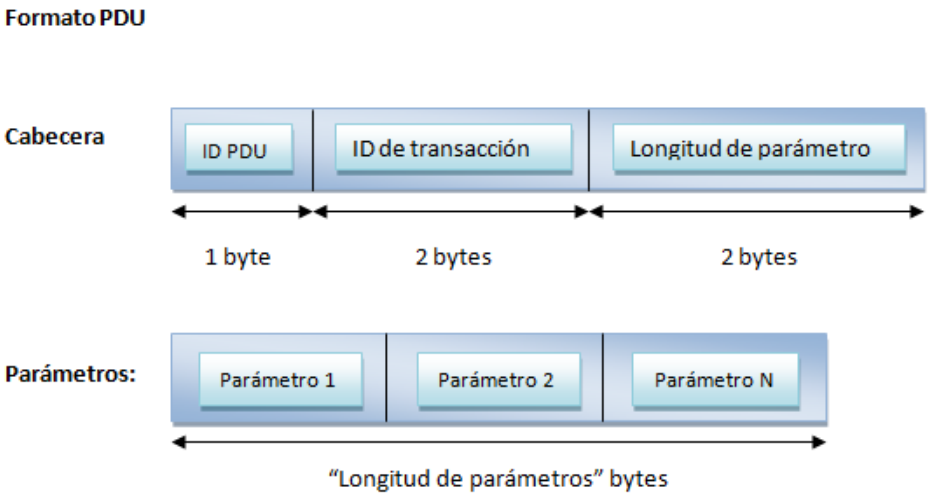


Figura 26: Formato de la PDU SDP.

- *ID PDU*: identifica el tipo de cada PDU.
- *ID de transacción*: identifica las PDUs que pertenecen a una misma solicitud y se emplea para emparejar las respuestas con las solicitudes.
- *Longitud de parámetros*: este campo especifica la longitud (en bytes) te todos los parámetros que forman la PDU.

Algunas solicitudes requieren respuestas que son más grandes que una única PDU de respuesta. En este caso, el servidor SDP genera una respuesta parcial junto con un parámetro que indica que faltan más respuestas. Este parámetro, es un campo de longitud variable cuyo primer byte indica el número de bytes adicionales que quedan por ser recibidos.

Después de que un cliente reciba una respuesta parcial y el correspondiente parámetro de continuación ya puede reeditar la solicitud original. Si el servidor determina que una solicitud no está bien formada o que por cualquier razón el servidor no puede responder con el tipo apropiado de PDU, responderá con una PDU *SDP_ErrorResponse*.

En la Tabla 4, podemos ver los diferentes tipos de PDUs que existen en SDP, con una breve descripción de cada una de ellas y los parámetros que tiene asociados.

ID	TIPO	DESCRIPCIÓN	PARÁMETROS
0X00	Reserved		
0X01	SDP_ErrorResponse	Respuesta a una PDU de solicitud inapropiada.	ErrorCode
0X02	SDP_ServiceSearchRequest	Localizar el registro de servicio que da el servicio de patrón de búsqueda como primer parámetro de	ServiceSearchPattern MaximunServiceRecordCount ContinuationState

		la PDU.	
0X03	SDP_ServiceSearchResponse	Respuesta a una solicitud válida SDP_ServiceSearchRequest.	TotalServiceRecordCount CurrentServiceRecordCount ServiceRecordHandleList ContinuationState
0X04	SDP_ServiceAttributeRequest	Recuperar los valores de atributos de un registro de servicio específico.	ServicerecordHandle MaximunAttributeByteCount AttributeIDList ContinuationState
0X05	SDP_ServiceAttributeResponse	Respuesta a una solicitud válida SDP_ServiceAttributeRequest.	AttributeListByteCount AttributeLists ContinuationState
0X06	SDP_ServiceSearchAttributeRequest	Combina las capidades de SDP_ServiceSearchRequest y SDP_ServiceAttributeRequest en una sola solicitud.	ServiceSearchPattern MaximunAttributeByteCount AttributeIDList ContinuationState
0X07	SDP_ServiceSearchAttributeResponse	Respuesta a una solicitud válida SDP_ServiceSearchAttributeRequest.	AttributeListByteCount AttributeLists ContinuationState
0X07 -0XFF	Reserved		

Tabla 4: Tipos de PDUs en SDP.

Los atributos universales son aquellos atributos de servicio cuyas definiciones son comunes a todos los registros de servicio:

- *Atributo ServiceRecordHandle*: es un número de 32 bits que identifica de manera única cada registro de servicio con un servidor SDP.
- *Atributo ServiceClassIDList*: consiste en una secuencia de Data Elements en la que cada uno de ellos es un UUID que representa las clases de servicio que forman un registro de servicio.
- *Atributo ServiceRecordState*: entero de 32 bits que se usa para facilitar el almacenamiento en caché de los atributos de servicio. Si este atributo aparece en un registro de servicio, su valor cambiará cuando se añada, elimine o cambie el valor de otro atributo.
- *Atributo ServiceID*: es un UUID que de manera única y universal identifica la instancia de servicio que es descrita por el registro de servicio.
- *Atributo ProtocolDescriptorList*: describe una o más pilas de protocolos que pueden usarse para ganar acceso al servicio descrito por el registro de servicio.

- *Atributo AdditionalProtocolDescriptorList*: contiene una secuencia de elementos del tipo *ProtocolDescriptorList*.
- *Atributo BrowseGroupList*: consiste en una secuencia de Data Elements en la que cada Data Element es un UUID que representa a un grupo de navegación al que pertenece el registro de servicio.
- *Atributo LanguageBaseAttributeIDList*: con el objetivo de soportar atributos que sean legibles por las personas en diferentes idiomas en un registro de servicio, se asigna un ID para cada idioma que se emplea en un registro de servicio.
- *Atributo ServiceInfoTimeToLive*: es un entero de 32 bits que representa el número de segundos que la información de un registro de servicio se supone que va a ser válida y no va a cambiar.
- *Atributo ServiceAvailability*: es un entero de 8 bits que representa la capacidad relativa del servicio aceptar clientes adicionales.
- *Atributo BluetoothProfileDescriptorList*: consiste en una secuencia de Data Elements en la que cada Data Element es un descriptor de perfil que contiene información sobre un perfil Bluetooth al que se ajusta el servicio representado por este registro de servicio.
- *Atributo DocumentatioURL*: este atributo es una URL que contiene la documentación del servicio descrito por el registro de servicio.
- *Atributo ClientExecutableURL*: contiene una URL que contiene la localización de una aplicación que puede usarse para utilizar el servicio descrito por el registro de servicio.
- *Atributo IconURL*: contiene una URL que hace referencia a la localización de un icono que se emplea para representar el servicio descrito por el registro de servicio.
- *Atributo ServiceName*: es una cadena de caracteres que contiene el nombre del servicio representado por el registro de servicio.
- *Atributo ServiceDescription*: es una cadena de menos de 200 caracteres que contiene una breve descripción del servicio.
- *Atributo ProviderName:* es una cadena de caracteres que contiene el nombre de la persona u organización que proporciona el servicio.

La clase de servicio "*ServiceDiscoveryServer*" describe un registro de servicio que contiene atributos del servidor de descubrimiento de servicios. Además de los atributos universales puede contener los siguientes:

- *Atributo ServiceRecordHandle*: descrito en la definición del atributo universal para el servicio *ServiceRecordHandle*.
- *Atributo ServiceClassIDList*: descrito en la definición del atributo universal para el servicio *ServiceClassIDList.*
- *Atributo VersionNumberList*: es una secuencia de Data Elements en la que cada uno de ellos es un número de versión soportado por el servidor SDP.
- *Atributo ServiceDatabaseState*: es un entero de 32 bits que se usa para facilitar el almacenamiento en caché de de los registros de servicio.

La clase de servicio "*BrowseGroupDescriptor*" describe el registro de servicio proporcionado por cada servicio *BrowseGroupDescriptor* ofrecido en un dispositivo Bluetooth. Además de los atributos universales puede contener los siguientes:

- *Atributo ServiceClassIDList*: descrito en la definición del atributo universal para el servicio *ServiceClassIDList*.
- *Atributo GroupID*: contiene un UUID que puede usarse para localizar servicios que son miembros de un grupo de navegación que este registro de servicio describe.

2.2.7.6 Nivel de Aplicación

Los procedimientos requeridos en la capa de aplicación son:

- *Establecimiento del enlace y configuración de la conexión serie virtual*: son los pasos necesarios para establecer la conexión en un dispositivo remoto para simular un puerto serie. Los pasos son:
    1. Enviar una solicitud usando SDP para encontrar el número del canal del servidor RFCOMM de la aplicación en concreto del dispositivo remoto.
    2. Opcionalmente, pedir la información necesaria para llevar a cabo la autenticación del dispositivo remoto. También, de manera opcional se puede solicitar encriptación.
    3. Solicitar un nuevo canal L2CAP a la entidad remota RFCOMM.
    4. Iniciar una sesión RFCOMM en el canal L2CAP.
    5. Comenzar una nueva conexión del enlace de datos en la sesión RFCOMM, empleando el número de canal del servidor mencionado anteriormente.

Después de este último paso, la conexión del cable serie virtual está listo para usarse para la comunicación entre aplicaciones.

- *Aceptación del enlace y establecimiento de la conexión serie virtual:* se pueden distinguir los siguientes pasos:
    1. Si es solicitado por el dispositivo remoto, se debe proporcionar la información necesaria para los procedimientos de autenticación y encriptación.
    2. Aceptar la indicación del establecimiento de un nuevo canal procedente de L2CAP.
    3. Aceptar el establecimiento de una sesión RFCOMM en este canal.
    4. Aceptar una nueva conexión del enlace de datos en la sesión RFCOMM. Esto puede provocar una solicitud local para autenticar el dispositivo remoto y activar la encriptación.
- *Anotación del servicio de registro en la base de datos local SDP:* este procedimiento se refiere al registro del servicio en la base de datos SDP. Implica la existencia de una base de datos de servicios y de la capacidad de responder a las *querys* del SDP. Todos los servicios/aplicaciones accesibles a través de RFCOMM necesitan suministrar un servicio de registro SDP que incluya los parámetros necesarios para poder localizar la aplicación/servicio correspondiente.

Para soportar las aplicaciones heredadas corriendo en puertos serie virtuales el servicio de registro lo debe realizar una aplicación auxiliar, la cual es añadida por el usuario cuando configura el puerto.

2.2.8 Dispositivos Bluetooth

En la actualidad, existen diversos tipos de dispositivos Bluetooth. Como por ejemplo, teléfonos móviles, consolas portátiles, impresoras, auriculares, relojes, etcétera. Como hemos mencionado en apartados anteriores, cada dispositivo Bluetooth tiene asignado una dirección IEEE MAC de 48 bits, conocida como dirección del dispositivo Bluetooth (BD_ADDR). Cada dispositivo contiene un campo llamado *Class of Device/Service* (en la cabecera del paquete de nivel banda base). Es un valor de 24 bits que el terminal envía junto con su dirección Bluetooth y que describe el tipo de dispositivo, así como el tipo de servicios que tiene disponibles. Como se puede observar en la Figura está compuesto de la siguiente manera:

- Los 2 primeros bits indican el campo *Format Type*, por defecto igual a 0.
- Los siguientes 11 bits están destinados al tipo de dispositivo (*Device Classes*):
    - Los 6 primeros bits están dedicados al *Minor Device Class*, campo que permite identificar el tipo específico de dispositivo.
    - Los 5 últimos bits representan al *Major Device Class*, el cual permite identificar el tipo genérico de dispositivo.
- Por último, los 11 bits restantes están reservados para el campo *Service Classes*, que describe los servicios soportados por el dispositivo.

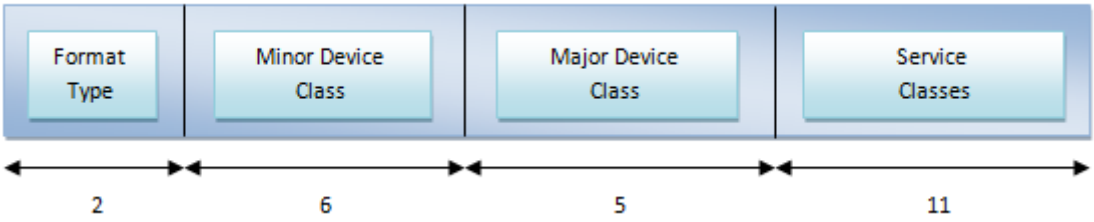


Figura 27: Estructura del campo *Class of Device/Service.*

En la siguiente tabla se pueden observar algunos tipos de *Major Device Classes* con algunos subtipos de *Minor Device Classes*:

MAJOR DEVICE CLASS	MINOR DEVICE CLASS
Computer (0x100)	Ordenador de sobremesa (0x104)
	Portátil (0x10C)
	PDA (0x114)
Phone (0x200)	Teléfono móvil (0x204)
	Smartphone (0x20C)
Wearable (0x700)	Reloj de pulsera (0x704)
	Casco (0x710)
	Gafas (0x714)

Tabla 5: Identificación de dispositivos Bluetooth.

En los apartados que vienen a continuación se describirán los dispositivos Bluetooth empleados a lo largo del proyecto.

1.1.1.1 2.2.8.1 WT12 (Placa de pruebas)

Es un módulo fabricado [Bluegiga 12] por la empresa *Bluegiga Technologies Inc*. que ofrece módulos inalámbricos basados en Bluetooth y soluciones de acceso a dispositivos para

OEMs, integradores y teleoperadores. Los dispositivos seguros, rentables y fáciles de integrar son utilizados por los líderes de la industria en las áreas de salud y medicina, aplicaciones de automoción, de audio, industrial y de consumo. Bluegiga fue fundada en el año 2000 con sede en Finlandia y oficinas de ventas en Estados Unidos y Hong Kong. A través de la red de distribución, Bluegiga está presente en más de 65 países de todo el mundo.

Figura 28: Logo Bluegiga [Bluegiga 12].

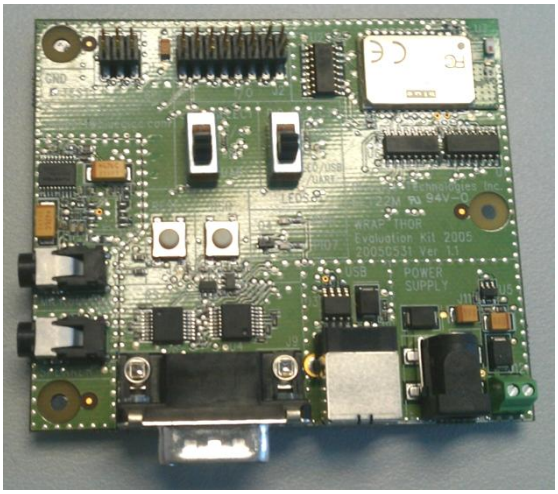


Figura 29: imagen de la placa de pruebas WT12.

WT12 es un módulo Bluetooth de clase 2 con Bluetooth 2.1 + EDR. Introduce [WT12 Data Sheet 10] [WT12 Product 09] una velocidad de datos tres veces superior a la ofrecida por la versión Bluetooth 1.2 que aparecían en módulos anteriores y el consumo de energía es menor. WT12 contiene integrados todos los elementos necesarios, desde una antena radio Bluetooth hasta una implementación completa de la pila de protocolos. Además este módulo es una solución ideal para los desarrolladores que quieren integrar la tecnología Bluetooth en sus diseños. Por defecto, el módulo WT12 está equipado con el firmware iWrap del que hablaremos en el próximo capítulo. También ofrece otras opciones de firmware, como son HCI (*Host Controller Interface*) y las interfaces USB o UART.

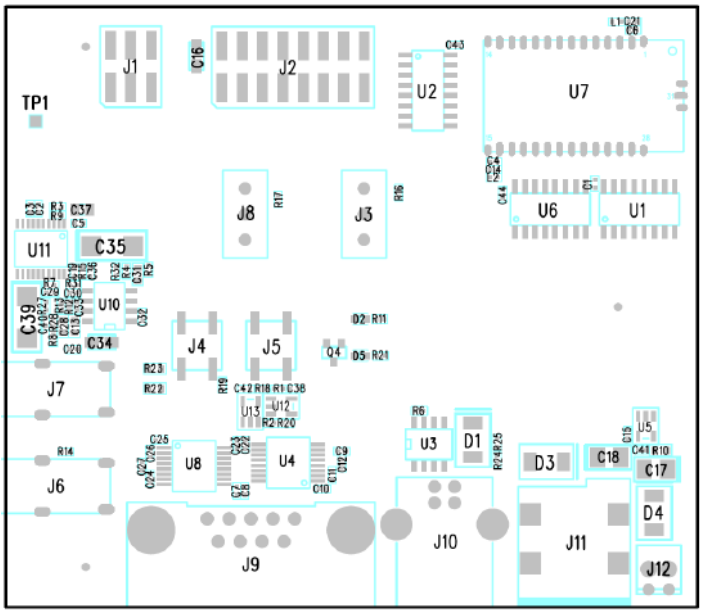


Figura 30: ensamblado de la placa de prueba WT12 [WT12 Data Sheet 10].

A continuación se resumen los principales elementos que componen el módulo WT12:

- *J1*: interfaz SPI, cuya interfaz física es una cabecera de pines 2x3 (AMP146134-2).
- *J2*: interfaz GPIO, cuya interfaz física es una cabecera de pines 2x8 (AMP146134-7).
- *J3*: selección PIO, este conmutador cambia la conexión de las señales PIO2 a PIO7 entre el conector J2 y las interfaces LED/USB/UART. Si el conmutador está en la parte superior, el módulo WT12 se comunica a través del conector J2 (GPIO). Si está en la parte inferior, el módulo emplea la interfaz a través del conector DB9 RS232.
- *J4*: pulsador de reset.
- *J5:* pulsador DRS. Está conectado al pin PIO5 y se usa si se quiere usar la señal DRS.
- *J6*: Jack audio, se usa para conectar unos altavoces o auriculares al módulo.
- *J7:* Jack voz, se usa para conectar un micrófono al módulo.
- *J8*: selección de SIG, este conmutador cambia la conexión de las señales nCTS y RxD entre el conector J2 y el conector DB9 RS232. Si el conmutador está en la parte superior, se usa cuando los pines nCTS y RxD externos al módulo WT12 se comunican a través del conector J2. Si está en la parte inferior, el módulo emplea la interfaz a través del conector DB9 RS232
- *J9:* interfaz DTE RS-232.
- *J10*: conector USB estándar.
- *J11*: fuente de alimentación, a través del cable de alimentación se proporciona a la placa los 5V necesarios para que funcione.

- *J12*: fuente de alimentación, se usa para proporcionar corriente a la placa de entre 5V y 9V a través de una fuente de alimentación externa.

6.2.1.1 WT32

Es la última generación de módulos Bluetooth del fabricante Bluegiga [WT32 Data 12] [WT32 Product 09]. Proporciona el mayor nivel de integración con radio integrada de 2,4 GHz, DSP, cargador de batería, códecs y una antena lista para aplicaciones de audio tanto mono como estéreo. WT32 está totalmente preparada para soportar el estándar Bluetooth 2.0. El núcleo DSP incorporado permite la mejora del producto con características como decodificación de audio avanzada, cancelación de eco, reducción del ruido y manipulación de datos. WT32 combinado con el firmware iWRAP, permite a los fabricantes añadir, de manera sencilla, conectividad Bluetooth en aplicaciones nuevas o ya existentes con un mínimo esfuerzo y desarrollo de fabricación. WT32 permite diferentes diseños de integración: el módulo puede ser configurado para que funcione de manera autónoma, un procesador de host puede controlar la funcionalidad Bluetooth con comandos ASCII a través de la interfaz UART, se pueden provocar funciones Bluetooth a través de eventos en los pines GPIO.

El módulo Bluetooth WT32 tiene disponible diferentes opciones de firmware para las comunicaciones de audio y datos. Bluegiga también proporciona el desarrollo de firmware personalizado de servicios de producción, eliminando la necesidad de los clientes para llevar a cabo un caro desarrollo de Bluetooth, certificaciones y la configuración de la línea de producción, reduciendo de manera significativa los costes de implementación, los riesgos y el tiempo.

El módulo WT32 ha sido la placa que se ha integrado con las motas SunSpot para proporcionarles tecnología Bluetooth en este proyecto. Su diagrama de bloques se muestra a continuación en la siguiente figura:

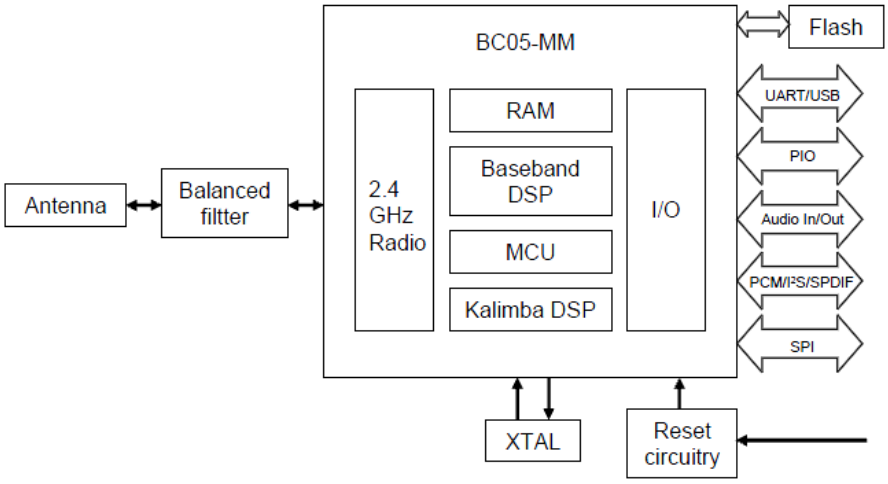


Figura 31: Diagrama de bloques del módulo WT32 [WT32 Data 12].

- *BC05-MM*: es un único chip que utiliza la interfaz radio y la banda base IC para sistemas Bluetooth en la banda de los 2,4 GHz. Cumple con el estándar Bluetooth 2.0 + EDR de la especificación de voz y datos.
- *Flash:* la memoria flash se utiliza para almacenar la pila de protocolos Bluetooth y aplicaciones de la máquina virtual. También se puede utilizar como una memoria RAM externa para aplicaciones que hacen un uso intensivo de la memoria.
- *Balanced Filter*: combina un balun y un filtro que cambia la señal balanceada de entrada/salida del módulo a una señal no balanceada de la antena. El filtro es un filtro paso banda (banda ISM).
- *Antenna*: WT32 usa una antena de chip cerámico con una constante dieléctrica alta, que hace que la antena sea muy insensible al ambiente que le rodea y por tanto, proporciona una libertad de diseño a su alrededor sin que le afecte.
- *USB:* es una interfaz de alta velocidad para comunicarse con otros dispositivos digitales compatibles. WT32 se comporta como un periférico USB, respondiendo a las solicitudes de un controlador del host principal, como por ejemplo un PC.
- *Synchronous Serial Interface*: es una interfaz de puerto serie síncrono (SPI) para interactuar con otros dispositivos digitales. El puerto SPI se puede usar para depurar el sistema. También se puede emplear para programar la memoria flash.
- *UART*: es una interfaz estándar transmisor-receptora asíncrona universal para comunicarse con otros dispositivos en serie.
- *PCM/I²S/SPDIF Interface*: es una interfaz serie, programable y bidireccional de audio que soporta los formatos PCM, I^2S y SPDIF.

- *Audio Interface*: la interfaz audio de WT32 tiene diferentes entradas y salidas y una salida para micrófono. Se puede implementar una aplicación de audio Bluetooth con una alta calidad estéreo con una cantidad mínima de componentes externos.
- *Programmable I/O*: WT32 tiene un total de 10 terminales entrada/salida digitales programables. Estos son controlados por el firmware del dispositivo.
- *Reset:* WT32 tiene un circuito para restablecer el módulo a una configuración de inicio y asegurar un funcionamiento adecuado de la memoria flash. Por otra parte, el reseteo se puede establecer de manera externa usando el pin de reset.
- *802.11 Coexistence Interface*: se proporciona un hardware dedicado para implementar una variedad de esquemas coexistentes. Soporta salto de canal AFH, señalización de prioridad, señalización de canal y e instrucciones de canal. Las características se configuran en el firmware.

A continuación, se describirá la interfaz UART proporcionada por el módulo WT32. Esta es la interfaz que se ha utilizado para conectar dicho módulo con la mota SunSpot. Esta interfaz proporciona un mecanismo simple para la comunicación entre dispositivos serie usando el estándar RS232. La UART de WT32, que puede verse en la Figura 32, emplea niveles de voltajes desde 0 a VDD_IO, necesitando un transceptor IC externo para proporcionar el voltaje necesario.

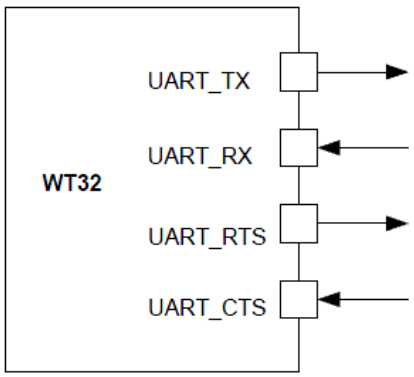


Figura 32: Interfaz UART WT32 [WT32 Data 12].

Se usan 4 señales para implementar las funcionalidades de la USART. Cuando WT32 se conecta a otro dispositivo, *UART_RX* y *UART_TX* transmiten datos entre los dos dispositivos. Las otras dos señales restantes, *UART_CTS* y *UART_RTS*, se pueden usar para implementar control de flujo a nivel de hardware. Las señales *DTR*, *DSR* y *DCD* se pueden implementar usando los pines *PIO* que hay en el módulo WT32. Todas las conexiones UART se implementan usando la tecnología CMOS.

A continuación, en la Tabla 6 se muestran los parámetros más destacados de la interfaz UART proporcionada por el módulo WT32.

PARÁMETRO	VALOR
Mínima velocidad de transmisión	1200 baudios
Máxima velocidad de transmisión	3 Mbaudios
Control de flujo	RTS/CTS, ninguno
Paridad	Ninguna, par o impar
Número de bits de parada	1 ó 2
Bits por canal	8

Tabla 6: Parámetros principales de la interfaz UART.

En el proyecto se ha utilizado una placa adaptadora comprada en Sparkfun (tienda online al por menor, que vende las partes y las piezas adaptados para proyectos de electrónica), donde el módulo WT32 ya tiene soldado todas las conexiones a una circuitería para poder usarla más cómodamente y poder conectar los cables.

Figura 33: placa adaptadora Sparkfun con el módulo WT32 integrado [Sparkfun 12].

1.1.1.2 2.2.8.3 Zephyr Bioharness v3

Está fabricado por la empresa Zephyr [Zephyr 12], la cuál es líder global en monitorización biomecánica y fisiológica en tiempo real o supervisión del estado físico (PSM), en soluciones para mHealth, en defensa y en los mercados de investigación. Fue fundad en 2003 y es una pequeña empresa con sede en Annapolis (Maryland, EEUU). Tiene una colaboración permanente con los departamentos de bomberos, la NASA y con las fuerzas especiales estadounidenses entre otros. Entre sus socios destacan Motorola y la NASA.

Zephyr

Figura 34: Logo de la empresa Zephyr [Zephyr 12].

El Bioharness 3 [Bioharness Data Sheet 11] es un módulo compacto de monitorización fisiológica. Está unido a una correa de tela ligera e inteligente ya que incorpora diferentes sensores. Puede transmitir datos fisiológicos vía Bluetooth o guardarlos en una memoria interna una vez medidos. Puede medir la tasa cardiaca, realizar un electrocardiograma (ECG), la tasa respiratoria, la aceleración, la postura, y la temperatura corporal, entre otros.

Figura 35: Zephyr Bioharness v3 [Bioharness User Manual 11].

Este dispositivo [Bioharness API 11] solo soporta una conexión Bluetooth empleando el perfil SPP y los parámetros velocidad de transmisión igual 115200 baudios, 8 bits de datos por canal, 1 bit de parada y sin paridad. El Bioharness utiliza un código de seguridad (1234), de esta manera cuando otro dispositivo quiera establecer una comunicación Bluetooth tendrá que introducir dicho código de seguridad para que se complete el establecimiento. Por defecto, si el Bioharness no recibe paquetes de datos durante 10 segundos se liberará la conexión. El cinturón Bioharness [Bioharness Specification 11] utiliza un mecanismo simple de transferencia de solicitud/respuesta (ACKs) para la mayoría de los paquetes que procesa, exceptuando los paquetes de datos que transportan flujos de datos. El Bioharness puede transmitir desde todos los paquetes existentes hasta ninguno de ellos. Esto se puede configurar a través de un software que incluye el dispositivo.

Existen 8 tipos de paquetes de datos diferentes:

- *Paquete de datos generales*: envía los datos cada 1s e incluye los siguientes parámetros:
    - *Tasa cardiaca.*
    - *Tasa respiratoria*
    - *Temperatura corporal*

  - *Actividad*
  - *Postura*
  - *Nivel de la batería*
- *Paquete de datos recopilados*: es una extensión del paquete de datos generales, incluyendo todos sus parámetros y otros muchos. Se puede configurar para que se envíe en un periodo de tiempo comprendido entre 1 – 65534 s. Más adelante, se describirá este paquete de manera más exhaustiva, ya que ha sido el que se ha usado para el proyecto.
- *Forma de la onda de la respiración*: contiene muestras de datos analógicos que se han tomado de un convertidor analógico-digital (sensor incluido en el cinturón) cada 56 ms. En recepción se puede formar una gráfica a partir de estas muestras y visualizar la forma de la onda de la respiración.
- *Datos R a R*: contiene muestras de periodos R-R, los cuáles se almacenan cada 56 ms. Un periodo R-R es el tiempo entre dos picos R de una gráfica ECG.
- *Forma de la onda ECG*: contiene muestras de datos analógicos que se han tomado de un convertidor analógico-digital cada 4 ms. En recepción se puede formar una gráfica a partir de estas muestras y visualizar el electrocardiograma.
- *Forma de la onda del acelerómetro*: contiene muestras de datos analógicos. De cada eje se han tomado muestras de un convertidor analógico-digital cada 20 ms. En recepción se puede formar una gráfica a partir de estas muestras y visualizar la forma de las ondas del acelerómetro (ejes X, Y, Z).
- *Eventos:* estos paquetes de datos se envían de manera asíncrona cuando tiene lugar un evento predeterminado. Hay cuatro categorías de eventos:
  - *Fisiológicos.*
  - *Eventos del sistema.*
  - *Eventos de error.*
  - *Eventos de depuración.*
- *Paquete para habilitar/deshabilitar comandos*: este tipo de paquete permite habilitar /deshabilitar, a través de comandos, algunos de los paquetes de datos mencionados anteriormente. En el caso de este proyecto, se ha empleado el comando "*Set Summary Data Packet Update Rate*" para habilitar el envío de paquetes de datos recopilados.

El formato del paquete general de la solicitud y de la respuesta se muestra en la siguiente Figura:

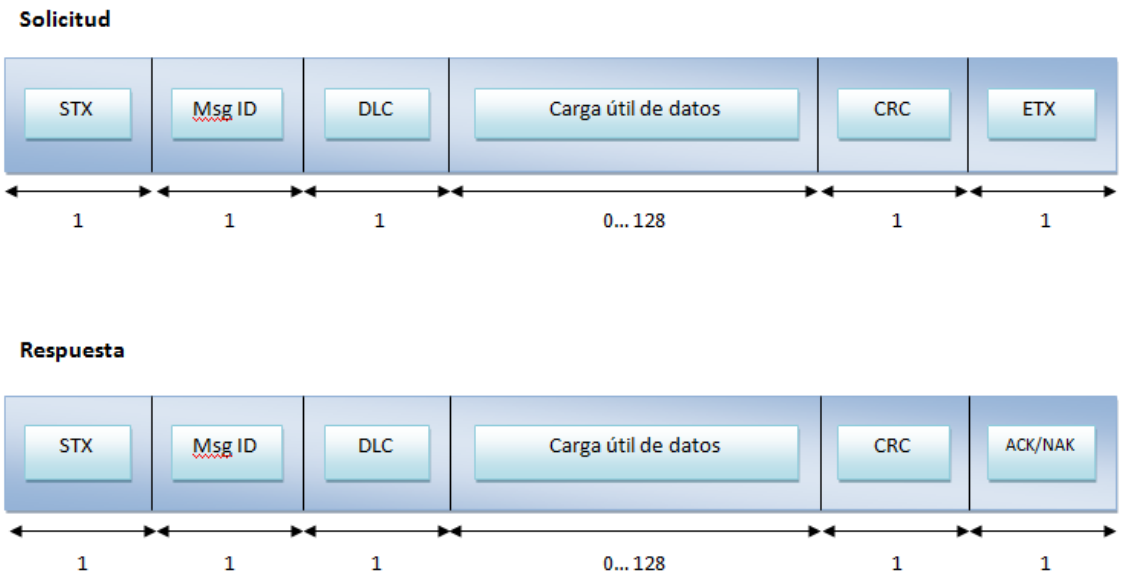


Figura 36: Formatos de las PDUs de solicitud y respuesta en el módulo Bioharness.

Los campos que lo forman son:

- *STX (1 octeto):* es un carácter de control estándar en ASCII (0x02) que indica el comienzo del mensaje.
- *Msg ID (1 octeto):* identifica de manera única cada mensaje. Un mensaje de respuesta usa el mismo Msg ID que la solicitud a la que está respondiendo.
- *DLC (1 octeto):* el código de longitud de datos se usa para especificar el número de bytes que viajan en la carga útil de datos.
- *Carga útil de datos (0 – 128 octetos):* contiene los datos que se envían entre las unidades local y remota.
- *CRC (1 octeto):* emplea un polinomio CRC-8 y solo tiene en cuenta la carga útil de datos para su cálculo.
- *ETX (1 octeto):* es un carácter de control estándar en ASCII (0x03) que indica el final del mensaje. Este campo solo aparece en el mensaje de solicitud, en la respuesta en su lugar se utiliza el campo ACK/NAK.
- *ACK/NAK (1 octeto):* este campo solo está presente en el mensaje de respuesta e indica si en el mensaje de solicitud se han encontrado errores o no. Si todos los datos son válidos en la respuesta se envía un ACK (0x06). Si por el contrario se ha producido algún error en la operación, se manda un NAK (0x15).

A continuación se describe el paquete de datos recopilados, que en nuestro caso es el que se ha utilizado en el proyecto. Aparece reflejado en la Figura 37.

En este caso, el campo *Msg ID* se corresponde con el valor *0x2B* y el campo *DLC* con el valor *71*. Este paquete tiene una longitud total de 76 bytes. Dentro del campo de carga útil nos interesan los bytes 13 y 14 ya que aquí se almacenan los bytes correspondientes a la tasa cardiaca. En los bytes 15 y 16 viajan los datos de la respiración y en el 64 y 65 la temperatura corporal. También cabe destacar los bytes 43 y 44 donde se almacena el valor del campo ROG. Este campo indica si la PDU se ha construido de manera correcta. Si no lo ha hecho, este paquete es descartado (*ROG = 000*).

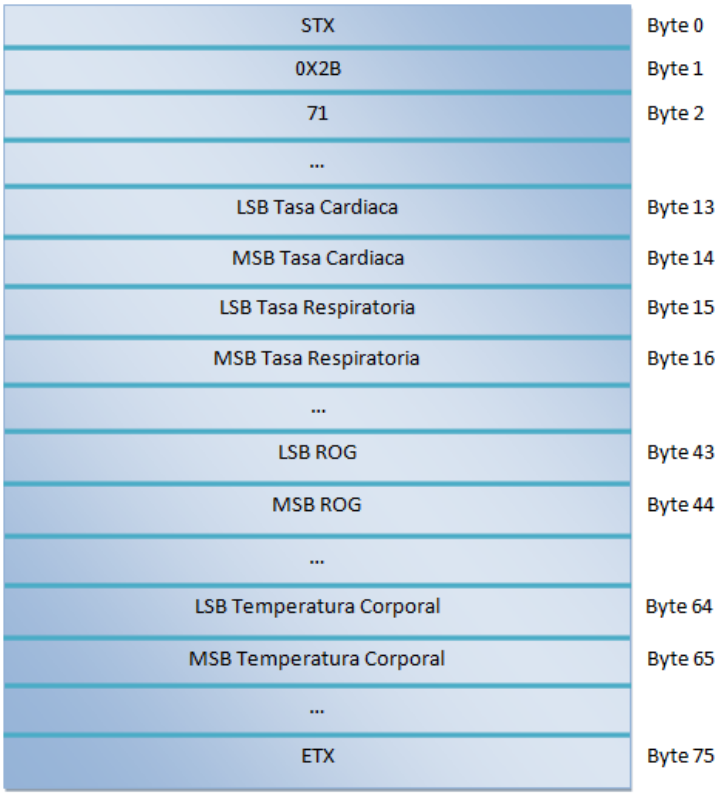


Figura 37: Representación de los campos utilizados del paquete de datos recopilados.

1.1.1.3 2.2.8.4 Reloj WiMM

WiMM Labs [WiMM 12] es una empresa con sede en Silicon Valley que ofrece el primer tipo de plataforma que permite que nuevos micro dispositivos inteligentes y llevables ofrezcan información de manera rápida, cómoda y sencilla.

Figura 38: Logo de la empresa WiMM Labs [WiMM 12].

Lo primero de todo, es configurar el módulo WiMM One. Para ello, se necesita una conexión Wi-Fi con acceso a Internet, un ordenador o dispositivo móvil con conexión a internet y un navegador web y el reloj Wimm.

A continuación, hay que crearse una cuenta en la web de WiMM para emparejar nuestro módulo con dicha cuenta. Desde esta cuenta podremos configurar todo lo que queramos en nuestro reloj, por ejemplo, los formatos de tiempo y fecha, la localización, el intervalo de sincronización de la cuenta WiMM con el dispositivo WiMM One, la pantalla del reloj con diferentes fondos de pantalla. También se pueden elegir las aplicaciones (micro apps) que queremos tener en nuestro reloj, como por ejemplo, un calendario, un cronómetro, información sobre el tiempo, un gestor de alarmas, etc. Además de las aplicaciones que diseñemos nosotros mismos.

Este dispositivo tiene tanto tecnología inalámbrica Wi-Fi como tecnología Bluetooth. Se puede elegir el sonido del dispositivo, si queremos que vibre o no y el brillo de la pantalla.

El WiMM One emplea la especificación Bluetooth 2.1 + EDR, soportando los perfiles SPP y de manos libres usando la pila RFCOMM de Android. El dispositivo solo se puede emparejar con un único dispositivo Bluetooth de manera manual, nunca de manera programada.

Figura 39: Reloj WiMM One.

Capítulo 3

Firmware iWRAP v4

3.1 INTRODUCCIÓN

iWRAP [iWRAP4 12] es un firmware embebido que corre en el procesador RISC de los módulos WT12, WT11, WT41 y WT32. Implementa la pila de protocolos Bluetooth completa y muchos perfiles Bluetooth. Todas las capas software, incluyendo el software de aplicación, corren en el procesador interno RISC en un entorno de ejecución software conocido como máquina virtual (VM).

El host puede acceder al firmware iWRAP a través de una o más interfaces, la interfaz más común es la UART que usa comandos ASCII para comunicarse con el host. De esta manera, el host puede acceder a las funcionalidades Bluetooth sin ninguna complejidad. La interfaz GPIO se puede usar para monitorizar eventos y para ejecutar comandos. Las interfaces PCM, SPDIF, I2S están disponibles para funciones de audio.

El usuario puede escribir una aplicación en el host para controlar el firmware iWRAP usando comandos ASCII o eventos GPIO. De esta manera, es fácil desarrollar aplicaciones para Bluetooth. En el módulo WT32 hay un procesador DSP extra para procesar los datos y el audio.

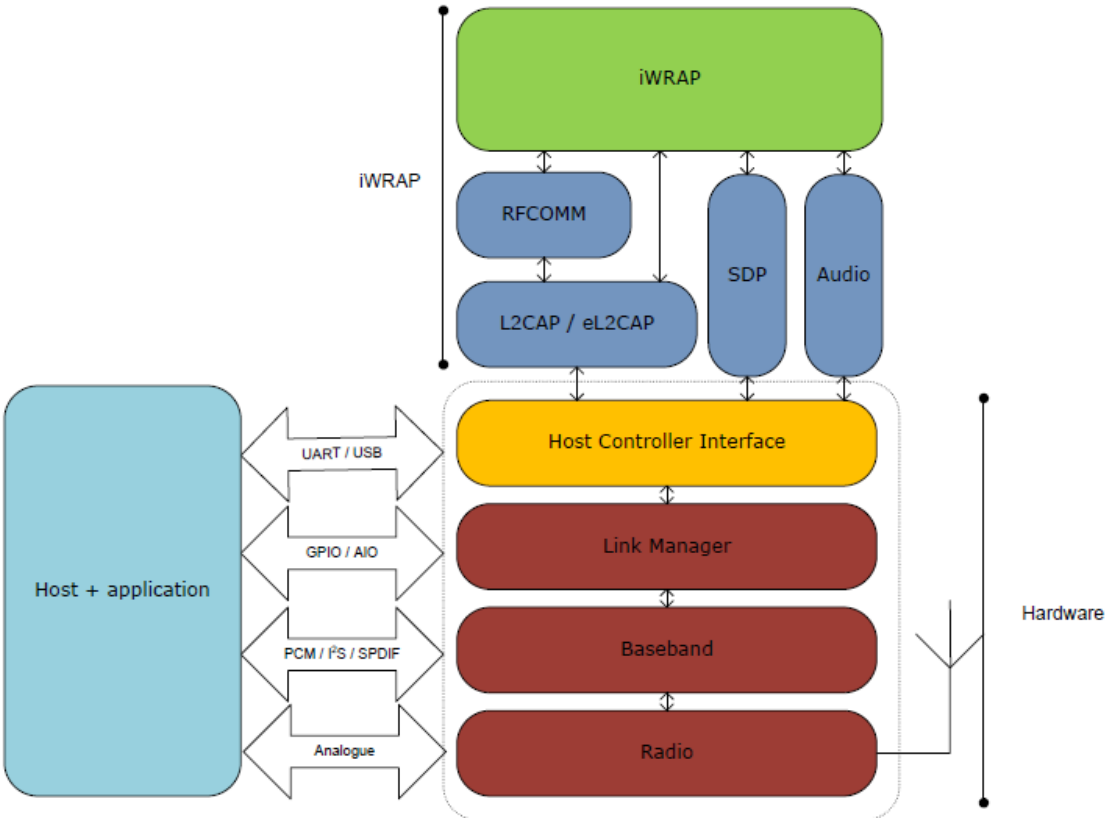


Figura 40: Pila Bluetooth implementada en el firmware iWRAP [iWRAP4 12].

Para empezar a usar el firmware iWRAP se puede usar un software terminal como el Hyperterminal o el software proporcionado por Bluegiga: BGTerm. Cuando se usa un software de terminal hay que asegurarse de que el módulo esté conectado al PC mediante el puerto serie. Por defecto, iWRAP usa la siguiente configuración en la interfaz UART:

- Velocidad de datos: 115200 bps.
- Bits de datos: 8.
- Bits de parada: 1.
- Bit de paridad: sin paridad.
- Control de flujo HW: activado.

3.2 MODOS IWRAP

En iWRAP existen dos modos de operación básicos:

- *Modo comando:* se le pasan comandos al firmware iWRAP para realizar acciones o cambios de configuración. Es el modo por defecto cuando no hay conexiones Bluetooth establecidas. La cantidad de datos que se pueden almacenar en los buffers depende de la versión firmware y del estado del iWRAP. Normalmente, suelen ser unos 1000 bytes.
- *Modo datos:* se emplea para transmitir/recibir datos a través del enlace Bluetooth. Solo se habilita si hay previamente una conexión Bluetooth establecida.

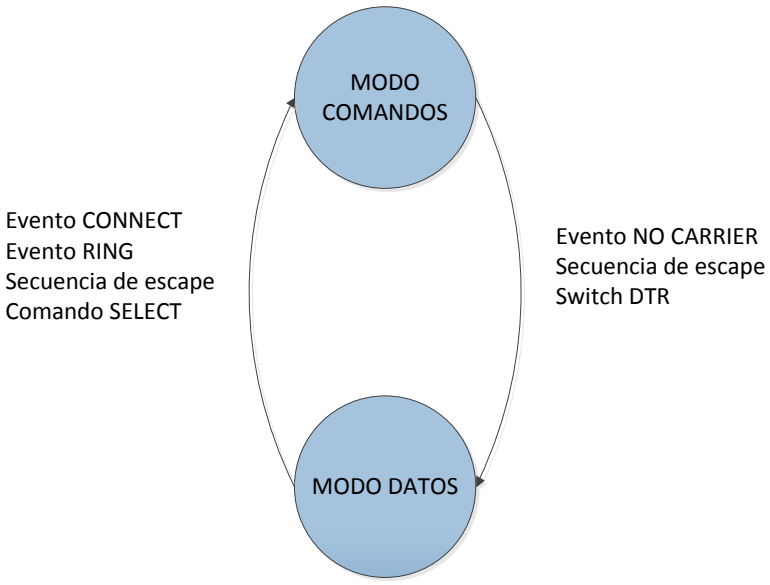


Figura 41: Transiciones de los modos iWRAP [iWRAP4 12].

La secuencia de escape en iWRAP permite el pasar del modo comando al modo de datos. La secuencia de escape, consiste en 3 caracteres de escape. Es definida por el comando SET CONTROL ESCAPE. Por defecto, el carácter de escape es '+'. Para que se reconozca como la secuencia de escape, no hay que introducir ni antes ni después de dicha secuencia ningún carácter durante un tiempo de guardia igual a 1s. Además, hay que introducir los caracteres de manera individual, no como una cadena de caracteres. Por defecto, esta es la secuencia de escape:

< Espera de 1 s > +++ < Espera de 1 s >

En la versiones igual a 2.1.0 o superiores existe un modo especial llamado modo de multiplexación. No distingue entre los modos de datos y comando, pero los datos, los comandos y los eventos se manejan en un único modo. La ventaja de este modo es que se pueden mantener conexiones simultáneas evitando hacer cada vez la transición de los estados a los modos datos-comandos-datos.

iWRAP distingue varios perfiles de conexión Bluetooth:

- *Perfil HFP*: es soportado a partir de la versión 2.2.0 iWrap. Se emplea como perfil de manos libres. Se emplea el modo comando para controlar todos los mensajes y eventos.
- *Perfil HSP*: sirve para establecer una conexión a través de unos auriculares.
- *Perfil OBEX*: iWRAP versión 4 soporta los modos OPP y FTP.
- *Perfil A2DP*: a partir de la versión 3 de iWRAP es un perfil de distribución de audio. Este perfil incluye mensajes y eventos de control.
- *Perfil AVRCP*: es soportado a partir de la versión 3 iWRAP. Es un perfil de control remoto de audio y video. También incluye mensajes y eventos de control, los cuales se manejan a través del modo comando.
- Perfil PBAP: es un perfil de acceso a la guía telefónica soportado en la versión 4 iWRAP. Emplea el modo comando para controlar mensajes y eventos.

3.3 COMANDOS AT

A continuación, se van a describir algunos de los comandos iWRAP más importantes y que se han empleado en este proyecto para la configuración de la placa de pruebas WT12 y de los dos módulos WT32.

- *Comando AT*: Confirma si el firmware de la placa está operativo.

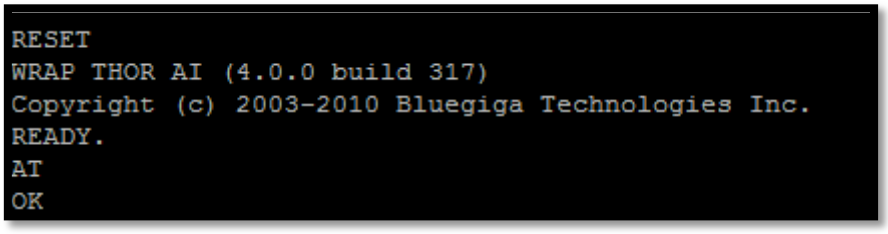


Figura 42: Ejemplo de comando AT en iWRAP.

- *Comando AUTH*: se puede usar para responde al evento AUTH para llevar a cabo el emparejamiento. Su sintaxis es *AUTH {bd_addr} [pin_code]*. No tiene ninguna respuesta. Ejemplo*: AUTH 00:07:80:81:66:8C 1234*.
- *Comando BATTERY:* se usa para leer el voltaje de la batería en mV. Solo funciona en el módulo WT32. Ejemplo:

  Comando: *BATTERY*

  Respuesta: *3673*
- *Comando BER*: Se emplea para consultar la tasa BER en el enlace. Su sintáxis es: *BER {id_enlace}* y la respuesta es: *BER {bd_addr} {ber}*. Ejemplo de uso en la Figura 43:

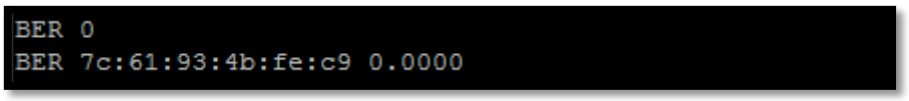


Figura 43: Ejemplo de comando VER en iWRAP.

- Comando *BYPASSUSART*: habilita el modo bypass de la UART, en el cual el tráfico se retransmite a los pines GPIO.
- *Comando CALL*: se emplea para establecer una conexión Bluetooth. Sintaxis: *CALL {address} {target} {connect_mode} [MTU {packet size}]*.
  - *Address:* dirección Bluetooth del dispositivo remoto.
  - *Target:* elegir entre RFCOMM, HFP, HFP-AG, HID o A2DP:
    - Número de canal.
    - Uuid 16.
    - Uuid 32.
    - Uuid 128.
    - L2CAP psm.
  - *Connect_mode*: define el modo de conexión que se va a establecer:
    - RFCOMM.

- HFP.
- HFP-AG.
- A2DP.
- AVRCP.
- HID.
- L2CAP.
- PBAP.
- OPP.
- FTP.
- HSP.
- HSP-AG.
- HDP.

  - *MTU*: parámetro opcional para indicar el tamaño del paquete que se va a usar.
  - *Packet size*: tamaño del paquete en bytes. Rango de 21 a 1008.

```
CALL 7c:61:93:4b:fe:c9 1105 OPP
CALL 0
CONNECT 0 OPP 2
OBEX 0 READY
```

Figura 44: Ejemplo del comando CALL en una conexión OPP.

- *Comando CLOCK*: Se emplea para conocer el valor del reloj de la red *piconet*. Es muy útil cuando los esclavos de dicha red tienen que sincronizarse. Estos esclavos se sincronizan con el reloj del maestro. La sintaxis del comando es: *CLOCK {link_id}*. Se puede ver un ejemplo en la Figura 45:

```
CLOCK 0
CLOCK 7c:61:93:4b:fe:c9 1362c6
```

Figura 45: Ejemplo del comando CLOCK en iWRAP.

- *Comando CLOSE*: Se emplea para cerrar la conexión Bluetooth. Su sintaxis es: *CLOSE {link_id}*. Por ejemplo:

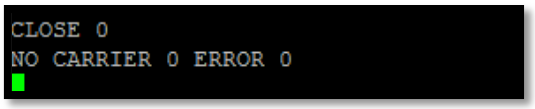
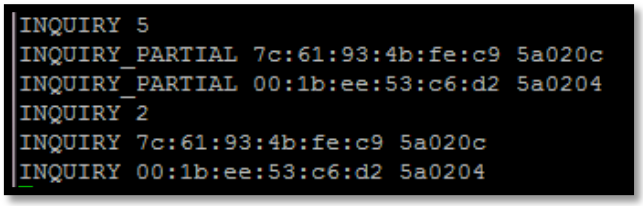


Figura 46: Ejemplo del comando CLOSE en iWRAP.

- *Comando CONNECT*: a partir de iWRAP 3.0, dicho firmware puede actuar como repetidor para conexiones RFCOMM usando el comando *CONNECT*. El cual, de manera transparente, puede enlazar dos conexiones en curso como una sola conexión entre los dos dispositivos remotos. La sintaxis del comando es: *CONNECT {link_id1} {link_id2}*.
- *Comando ECHO*: envía una cadena específica de caracteres al enlace activo especificado por el parámetro *link_id*. Este comando se puede utilizar junto al comando *SET CONTROL BIND* para enviar una indicación de la actividad que hay sobre un enlace Bluetooth. Su sintaxis es: *ECHO {link_id} [string].*
- *Comando INQUIRY:* comando empleado para descubrir otros dispositivos visibles en su rango de alcance. Su sintaxis es: *INQUIRY {timeout} [NAME] [LAP {lap}]*:
    - *Timeout:* el máximo tiempo que puede transcurrir antes de que acabe el proceso de inquiry.
    - *NAME:* flag opcional para que se muestre el "friendly name" (nombre dado al dispositivo por parte del usuario) al encontrar dispositivos.
    - *LAP:* flag opcional para especificar el uso de un código de acceso.
    - *Lap:* valor del código de acceso.

Aquí dos ejemplos (Figura 47):



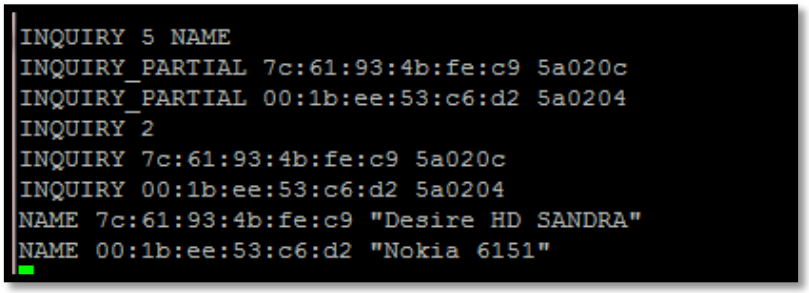


Figura 47: Ejemplos del comando INQUIRY. En el segundo caso, con la opción NAME.

Se puede observar cómo se produce el evento *INQUIRY PARTIAL*, el cual ocurre cuando se descubre un dispositivo. Finalmente, se hace un resumen de todos los dispositivos descubiertos. La respuesta tiene el formato *INQUIRY {bd_addr} {class_of_device} [NAME {bd_addr} {friendly-name}]*.

- *Comando IC*: (inquiry cancel) se emplea para cancelar una búsqueda de dispositivos que ha sido iniciada.
- *Comando IDENT*: se emplea para identificar un dispositivo Bluetooth remoto con el ID del perfil del dispositivo Bluetooth. Su sintaxis es *IDENT {bd_addr}.*
- *Comando KILL*: se usa explícitamente para terminar con todas las conexiones ACL entre dos dispositivos. Su sintaxis es: *KILL {bd_addr} [reason]*. El parámetro reason indica el motivo para desconectar el enlace ACL.
- *Comando L2CAP*: se usa para crear un canal PSM L2CAP para coneciones en el dispositivo local. La sintaxis es *L2CAP {psm}*.
- *Comando LIST*: Se emplea para conocer las conexiones Bluetooth existentes. Primero indica cuántas conexiones hay y después un resumen de cada una de ellas.

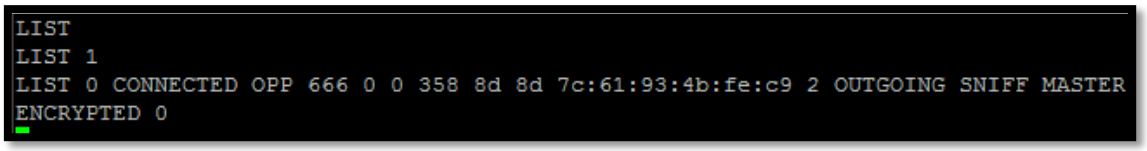


Figura 48: Ejemplo del comando LIST en iWRAP.

Como se observa en el ejemplo, existe una conexión establecida con las siguientes características:

- Número de conexiones: 1
- Identificador del enlace: 0
- Modo: OPP conectado
- Tamaño del paquete de datos: 666
- Tiempo en segundos que lleva establecido el enlace: 358
- Control del estado del modem en el puerto serie local: 8d
- Control del estado del modem en el puerto serie remoto: 8d
- Dirección BT del dispositivo remoto: 7c:61:93:4b:fe:c9
- Número de canal : 2
- Dirección: OUTGOING (La conexión ha sido inicializada por el dispositivo local).
- Modo de alimentación: SNIF
- Rol: Máster

- Estado de la encriptación de la conexión: ENCRYPTED.
- Datos almacenados en el buffer de entrada: 0

- *Comando NAME*: se utiliza para descubrir el "*friendly name*" del dispositivo. Su sintaxis es: *NAME {bd_addr}*, en la respuesta aparecerá el "*friendly-name*".
- Comando PAIR: Se emplea para saber si hay una conexión establecida con el dispositivo que se específica. Su sintaxis es *PAIR {bd_addr}.*

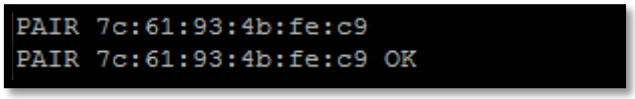


Figura 49: Ejemplo del comando PAIR en iWRAP.

- *Comando PING*: Se hace un ping indicando el ID del enlace. Se devuelve el tiempo (ms) de ida y vuelta.

Figura 50: Ejemplo del comando PING en iWRAP.

- *Comando RFCOMM*: se usa para crear un canal RFCOMM para conexiones generales RFCOMM. Su sintaxis es *RFCOMM {action}*, donde *action* es igual a *CREATE*. La respuesta sería *RFCOMM {channel}.*
- *Comando RESET*: se utiliza para hacer un reset a nivel software.
- *Comando RSSI*: Indicando el Id del link, nos muestra la potencia con la que se recibe la señal (dB).

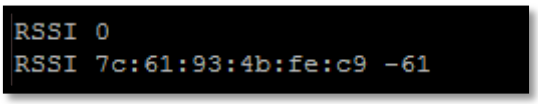


Figura 51: Ejemplo del comando RSSI en iWRAP.

- *Comando SDP*: Se emplea para navegar por los servicios disponibles en otros dispositivos Bluetooth. Su sintaxis es *SDP {bd_addr} {uuid} [ALL]*, donde:
  - *Bd_addr*: es la dirección del dispositivo remoto Bluetooth.
  - *Uuid*: indica el servicio a buscar para obtener información.
  - *ALL:* flag opcional para leer toda la información SDP del dispositivo remoto.

Una posible respuesta podría tener el siguiente formato: *SDP {bd_addr} < I SERVICENAME S "service_name" > < I PROTOCOLDESCRIPTORLIST < < U L2CAP psm> < U RFCOMM I channel > > >*

```
SDP 7c:61:93:4b:fe:c9 1105
SDP 7c:61:93:4b:fe:c9 < I SERVICENAME S "Object Push" > < I PROTOCOLDESCRIPTORLIS
T < < U L2CAP > < U RFCOMM I 03 > < U OBEX > > >
SDP
```

Figura 52: Ejemplo del comando SDP y su respuesta correspondiente en iWRAP.

- *Comando SDP ADD*: se usa para modificar el registro del servicio local para añadir servicios nuevos basados en RFCOMM. Es útil cuando se quiere usar un perfil Bluetooth que iWRAP no soporta. Su sintaxis es: *SDP ADD {uuid} {name}*. La respuesta será *SDP {channel}*.
- *Comando SELECT*: se utiliza para cambiar del modo comando al modo de datos. Su sintaxis es *SELECT {link_id}*.
- *Comando SET*: con este comando se puede visualizar o configurar diferentes parámetros y configuraciones en el firmware iWRAP. Dichos ajustes se guardan en una memoria no volátil, por lo que solo hay que configurarlos una vez. Su sintaxis es: *SET [{category} {option} {value}]:*
    - *Category*: categoría de la configuración:
        - *BT*: cambios relacionados con la configuración Bluetooth.
        - *CONTROL*: cambios relacionados con la configuración iWrap.
        - *PROFILE*: activa o desactiva perfiles Bluetooth.
        - *Link_id*: cambios relacionados con la configuración de los enlaces Bluetooth en iWrap.
    - *Option*: nombre de la opción, que depende de la categoría.
    - *Value*: valor de la opción.

```
SET
SET BT BDADDR 00:07:80:4a:23:3f
SET BT NAME WT12-A
SET BT CLASS 001f00
SET BT AUTH * 1234
SET BT IDENT BT:47 f000 4.0.0 Bluegiga iWRAP
SET BT LAP 9e8b33
SET BT PAGEMODE 4 2000 1
SET BT PAIR 7c:61:93:4b:fe:c9 07b921f4ddb6fb60c614716a2b7310e2
SET BT POWER 3 3 3
SET BT ROLE 0 f 7d00
SET BT SNIFF 0 20 1 8
SET BT MTU 667
SET CONTROL BAUD 115200,8n1
SET CONTROL CD 00 0
SET CONTROL ECHO 7
SET CONTROL ESCAPE 43 00 1
SET CONTROL GAIN 0 5
SET CONTROL MSC DTE 00 00 00 00 00 00
SET CONTROL PREAMP 1 1
SET CONTROL READY 00
SET PROFILE SPP Bluetooth Serial Port
SET
```

Figura 53: Ejemplo del comando SET en iWRAP.

- *Comando SET BT AUTH*: muestra o configura el código PIN del dispositivo local. En el siguiente ejemplo se configura el pin 1234 para que todos los dispositivos que quieran establecer comunicación con dicho dispositivo se tengan que autenticar:

```
SET BT AUTH * 1234
```

Figura 54: Ejemplo del comando SET BT AUTH.

- *Comando SET BT CLASS*: establece la clase de dispositivo local Bluetooth (CoD). La clase de dispositivo es el parámetro que se recibe durante el descubrimiento de dispositivos, indicando el tipo de dispositivo y que servicios soporta. Su sintaxis es *SET BT CLASS {class_of_device}*.
- *Comando SET BT MTU*: configura el tamaño máximo del paquete para las conexiones Bluetooth RFCOMM. Su sintaxis es: *SET BT MTU {mtu}*. El valor de *mtu* es entre 22 y 1009.
- *Comando SET BT NAME*: configura el "*friendly name*" del dispositivo local. Su sintaxis es: *SET BT NAME {friendly_name}*.
- *Comando SET BT PAIRCOUNT*: se emplea para establecer el número máximo de emparejamientos (entre 0 y 16) que aceptará iWRAP. Su sintaxis es *SET BT PAIRCOUNT {max_pairings}*.

- *Comando SET BT PAIR*: muestra o configure la información de emparejamiento del dispositivo local. Su sintaxis es: *SET BT PAIR {bd_addr} {link_key}.*
- *Comando SET BT POWER*: cambia los parámetros de potencia del módulo Bluetooth. Su sintaxis es: *SET BT POWER [RESET]| [{default} {maximun} [inquiry]]*. Si no se le pasan parámetros, simplemente muestra los valores configurados de potencia. A continuación, se describen los parámetros del comando:
    - *RESET:* vuelve a los valores por defecto configurados en la potencia Tx y resetea iWrap.
    - *Default:* potencia por defecto en Tx en dBm.
    - *Maximum:* máxima potencia Tx en dBm.
    - *Inquiry*: transmite la potencia en dBm usada para la operación INQUIRY.
- *Comando SET BT ROLE*: con este comando se puede mostrar o configurar el rol que ejerce el dispositivo local (maestro o esclavo). Su sintaxis es: *SET BT ROLE {ms_policy} {link_policy} {supervisión_timeout}.*
    - *Ms_policy*: este parámetro define como se comporta la función policía maestro-esclavo.
        - *0*: este valor permite que el maestro-esclavo cambie cuando se hace una llamada, pero iWRAP no lo pide al contestar (valor por defecto).
        - *1*: este valor permite que el maestro-esclavo cambie cuando se hace una llamada, e iWRAP lo pide al contestar.
        - *2:* no permite que el maestro-esclavo cambie cuando se realiza una llamada.
    - *Link_policy*:
        - *Bit 1*: si se activa, se habilita el cambio de rol.
        - *Bit 2*: si se activa, se habilita el modo Hold.
        - *Bit 3*: si se activa, se habilita el modo Sniff.
        - *Bit 4*: si se activa, se habilita el estado Park.
        - *F y 0*: estos valores habilitan todos los modos posibles (valor por defecto).
    - *Supervision_timeout*: controla cuanto tiempo se mantiene abierto un enlace Bluetooth si no se recibe ninguna respuesta por parte del dispositivo remoto.
- *Comando SET BT SSP*: este comando habilita/deshabilita el modo SSP del Bluetooth 2.1 + EDR. Su sintaxis es: SET *BT SSP {capabilities} {mitm}*.
    - *Capabilities*:

- *0*: solo mostrar.
- *1*: mostrar + botón si/no.
- *2*: solo teclado.
- *3*: ninguno.

- *Mitm*:
  - *0*: deshabilitar la protección man-in-the-middle.
  - *1*: habilitar la protección man-in-the-middle.

Figura 55: Ejemplo del comando SET BT SSP.

- *Comando SET CONTROL AUTOCALL*: habilita/deshabilita la funcionalidad de autollamada en iWrap. Si está habilitado, iWRAP intenta conectarse al dispositivo que está emparejado. Su sintaxis es: *SET CONTROL AUTOCALL {target} {timeout} {profile*}.
  - *Target:* elegir entre RFCOMM, HFP, HFP-AG, HID o A2DP:
    - *Número de canal.*
    - *Uuid 16.*
    - *Uuid 32.*
    - *Uuid 128.*
    - *L2CAP psm.*
  - *Timeout*: tiempo de espera entre llamadas (ms).
  - *Profile*: define el modo de establecer una conexión
    - *RFCOMM.*
    - *HFP.*
    - *HFP-AG.*
    - *A2DP.*
    - *AVRCP.*
    - *HID.*
    - *L2CAP.*
- *Comando SET CONTROL BAUD*: cambia la configuración de los parámetros de la UART. Su sintaxis es: *SET CONTROL BAUD {baud_rate},8 {parity} {stop_bits}:*
  - *Baud_rate*: velocidad de datos en bps.
  - *Parity:* paridad de la UART:
    - *N*: sin paridad.

        - *E*: paridad par.
        - *O*: paridad impar.
    - *Stop_bits*: número de bits de parada.
        - *1:* un bit de parada.
        - *2*: dos bits de parada.

Ejemplo: *SET CONTROL BAUD 115200,8N1*.

- *Comando SET CONTROL INIT*: muestra o permite cambiar un comando al inicializar iWRAP. Su sintaxis es: *SET CONTROL INIT [command].*
- *Comando SET CONTROL MUX*: se usa para habilitar/deshabilitar el modo de multiplexaicón. Su sintaxis es: *SET CONTROL MUX {mode}*, donde *mode* puede ser:
    - *0*: se deshabilita el modo multiplexación y se habilita el modo normal (datos-comandos).
    - *1*: se habilitado el modo de multiplexación.
- *Comando SET CONTROL MSC*: permite transmitir todas las señales UART sobre el el perfil SPP del enlace Bluetooth. Su sintaxis es: *SET CONTROL MSC [[mode] [[DSR] [[DTR] [[RTS] [[CTS] [[RI] [DCD]]]]]]*.
    - *Mode*: modo del dispositivo que se conecta a iWRAP. Puede ser DTE ó nDTE y DCE ó nDCE.
- *Comando SET {link_id} ACTIVE*: este comando deshabilita todos los modos de ahorro de energía, activa el enlace Bluetooth y entrar en el modo activo. Por ejemplo:

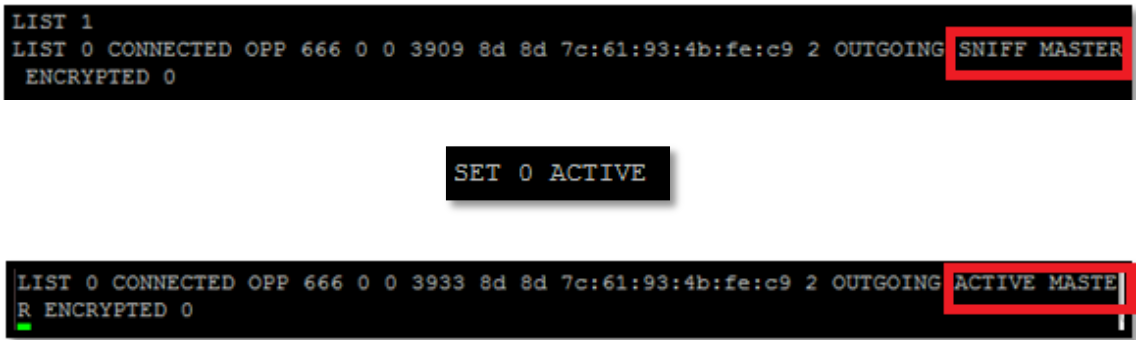


Figura 56: Uso del comando SET {link_id} ACTIVE.

- *Comando SET {link_id} MASTER*: intenta cambiar el enlace al maestro de la *piconet*.
- *Comando SET {link_id} SLAVE*: intenta cambiar el enlace al esclavo de la *piconet*.
- *Comando SET {link_id} SELECT*: con este comando se puede definir la conexión activa Bluetooth. Este comando es útil cuando, por ejemplo, hay dos conexiones simultáneas y se quiere elegir una de ellas.

- *Comando SET PROFILE*: se usa para activar o desactivar los perfiles Bluetooth. Su sintaxis es: *SET PROFILE {profile_name} [sdp_name]:*
    - *Profile_name*: indica el perfil específico a activar/desactivar. Los perfiles son:
        - *SPP.*
        - *HFP.*
        - *HFP-AG.*
        - *OPP.*
        - *A2DP SINK.*
        - *A2DP SOURCE.*
        - *AVRCP CONTROLLER.*
        - *AVRCP TARGET.*
        - *HID.*
        - *HSP.*
        - *HSP-AG.*
        - *HDP SINK.*
        - *HDP SOURCE.*
        - *PBAP.*
        - *BGIO.*
        - *OTA.*
    - *Sdp_name*:
        - *ON*: habilita el perfil con el nombre por defecto SDP.
        - *<string>:* habilita el perfil con el nombre del string (máximo 49 caracteres).

  Ejemplo: *SET PROFILE SPP ON.*

- *Comando SET RESET*: permite que el módulo vuelva a los valores de fábrica.

```
SET RESET
WRAP THOR AI (4.0.0 build 317)
Copyright (c) 2003-2010 Bluegiga Technologies Inc.
READY.
```

Figura 57: Utilización del comando SET RESET en iWRAP.

- *Comando SLEEP*: fuerza el modo *SLEEP*. Después de mandar este comando, el módulo entrará en el modo *SLEEP* hasta que se reciba una conexión Bluetooth o bien se reciba algo desde la interfaz UART.
- *Comando SSP CONFIRM*: se usa para confirmar o cancelar las solicitudes SSP de otros dispositivos Bluetooth. La sintaxis es la siguiente: *SSP CONFIRM {bd_addr} [OK].*

3.4 ERRORES SDP

A continuación, se mostrará una tabla con los posibles errores SDP que pueden ocurrir, el código y su descripción. Todo los códigos comienzan por 0x300.

MENSAJE DE ERROR	CÓDIGO	DESCRIPCIÓN
SDC_OK	0x00	-
SDC_OPEN_SEARCH_BUSY	0x01	El registro SDP está ocupado.
SDC_OPEN_SEARCH_FAILED	0x02	El registro SDP ha fallado.
SDC_OPEN_SEARCH_OPEN	0x03	-
SDC_OPEN_DISCONNECTED	0x04	-
SDC_OPEN_SEARCH_FAILED_PAGE_TIMEOUT	0x05	El registro SDP ha fallado debido al tiempo de espera en la paginación.
SDC_OPEN_SEARCH_FAILED_REJ_PS	0x06	-
SDC_OPEN_SEARCH_FAILED_REJ_SECURITY	0x07	El registro SDP ha fallado por motivos de seguridad.
SDC_OPEN_SEARCH_FAILED_REJ_RESOURCES	0x08	El registro SDP ha fallado debido a la insuficiencia de recursos.
SDC_OPEN_SEARCH_FAILED_SIGNAL_TIMEOUT	0x09	-
SDC_ERROR_RESPONSE_PDU	0x10	-
SDC_NO_RESPONSE_DATA	0x11	Respuesta vacía – sin resultados.
SDC_CON_DISCONNECTED	0x12	Dispositivo remote desconectado
SDC_CONNECTION_ERROR	0x13	El dispositivo remoto ha rechazado la conexión.
SDC_CONFIGURE_ERROR	0x14	La configuración L2CAP ha fallado.
SDC_SEARCH_DATA_ERROR	0x15	La búsqueda de datos es inválida.
SDC_DATA_CFM_ERROR	0x16	Fallo al transmitir una PDU.
SDC_SEACRH_BUSY	0x17	La búsqueda está ocupada.
SDC_RESPONSE_PDU_HEADER_ERROR	0x18	La respuesta ha tenido un error en la cabecera.
SDC_RESPONSE_PDU_SIZE_ERROR	0x19	La respuesta ha tenido un error de tamaño.
SDC_RESPONSE_TIMEOUT_ERROR	0x1a	La respuesta ha expirado.
SDC_SEARCH_SIZE_TOO_BIG	0x1b	El tamaño de la búsqueda no

		cabe dentro del paquete L2CAP.
SDC_RESPONSE_OUT_OF_MEMORY	0x1c	-
SDC_RESPONSE_TERMINATED	0x1d	-
SDC_OPEN_SEARCH_FAILED_PAGE_TIMEOUT	305	La búsqueda SDP ha fallado a causa del tiempo de espera en la paginación.
SDC_OPEN_SEARCH_FAILED_REJ_TIMEOUT	305	La búsqueda SDP ha fallado a causa del tiempo de espera en la paginación.

Tabla 7: Mensajes de error SDP en iWRAP.

3.5 ERRORES RFCOMM

A continuación, se muestra una tabla con los posibles errores RFCOMM que pueden ocurrir. Todos los códigos de este tipo empiezan por 0x400.

MENSAJE DE ERROR	CÓDIGO
RFC_OK	0x00
RFC_CONNECTION_PENDING	0x01
RFC_CONNECTION_REJ_PSM	0x02
RFC_CONNECTION_REJ_SECURITY	0x03
RFC_CONNECTION_REJ_RESOURCES	0x04
RFC_CONNECTION_REJ_NOT_READY	0x05
RFC_CONNECTION_FAILED	0x06
RFC_CONNECTION_TIMEOUT	0x07
RFC_NORMAL_DISCONNECT	0x08
RFC_ABNORMAL_DISCONNECT	0x09
RFC_CONFIG_UNACCEPTABLE	0x0a
RFC_CONFIG_REJECTED	0x0b
RFC_CONFIG_INVALID_CID	0x0c
RFC_CONFIG_UNKNOWN	0x0d
RFC_CONFIG_REJECTED_LOCALLY	0x0e
RFC_CONFIG_TIMEOUT	0x0f
RFC_REMOTE_REFUSAL	0x11
RFC_RACE_CONDITION_DETECTED	0x12
RFC_INSUFFICIENT_RESOURCES	0x13
RFC_CANNOT_CHANGE_FLOW_CONTROL_MECHANISM	0x14
RFC_DLC_ALREADY_EXISTS	0x15
RFC_DLC_REJ_SECURITY	0x16
RFC_GENERIC_REFUSAL	0x1f
RFC_UNEXPECTED_PRIMITIVE	0x20
RFC_INVALID_SERVER_CHANNEL	0x21
RFC_UNKNOWN_MUX_ID	0x22
RFC_LOCAL_ENTITY_TERMINATED_CONNECTION	0x23
RFC_UNKNOWN_PRIMITIVE	0x24

RFC_MAX_PAYLOAD_EXCEEDED	0x25
RFC_INCONSISTENT_PARAMETERS	0x26
RFC_INSUFFICIENT_CREDITS	0x27
RFC_CREDIT_FLOW_CONTROL_PROTOCOL_VIOLATION	0x28
RFC_RES_ACK_TIMEOUT	0x30
RFC_CONNECTION_REJ_SSP_AUTH_FAIL	-
L2CAP_CONNECTION_SSP_AUTH_FAIL	-

Tabla 8: Mensajes de error RFCOMM en iWRAP.

3.6 EJEMPLO DE CONEXIÓN BLUETOOTH EMPLEANDO EL PERFIL SPP

A continuación se explicarán todos los pasos para configurar un perfil SPP en la placa WT12 [iWRAP SPP 10] y establecer una comunicación Bluetooth. Tanto ejerciendo el rol de "Iniciador" de la llamada como "aceptador" de la llamada.

3.6.1 Llamada saliente WT12 (Iniciador)

Comenzaremos con el caso de que el módulo WT12 actúe como "iniciador" de la comunicación (maestro).

Lo primero de todo es comprobar que la placa está encendida y funcionando correctamente. Para ello, lo comprobamos con el comando *AT*. A continuación, activamos el perfil SPP en el módulo mediante el comando *SET PROFILE SPP ON*. De manera opcional, se puede establecer una clave de autenticación con el comando *SET BT AUTH * 1234*. Ahora reiniciamos el módulo con el comando *RESET* para que se guarden los cambios que hemos realizado. Podemos comprobar si se han realizado dichos cambios mediante el comando *SET*.

El paso siguiente es hacer una búsqueda de dispositivos Bluetooth mediante el comando *INQUIRY*, en nuestro caso haremos una búsqueda de 5 s conociendo el "*friendly name*" de cada dispositivo que encontremos. El comando en concreto será *INQUIRY 5 NAME*. Una vez encontrado el dispositivo al que nos queremos conectar, tendremos que activar en el dispositivo remoto el puerto serie virtual. Una vez hecho esto, realizamos la llamada Bluetooth para conectarnos al dispositivo remoto, en nuestro caso, con el comando *CALL 00:1A:92:7A:92:37 1101 RFCOMM.*

```
CALL 00:1a:92:7a:92:37 1101 RFCOMM
CALL 0
CONNECT 0 RFCOMM 1
```

Figura 58: Ejemplo del establecimiento correcto de una llamada.

Si la contestación, por parte del dispositivo remoto es *CALL 0*, nos indica que el id del enlace es el 0 y a continuación aparece el evento *CONNECT 0 RFCOMM 1*, lo cual indica que se ha establecido correctamente la conexión y que se ha asignado el canal 1 RFCOMM para el intercambio de datos.

Comprobamos que podemos enviar datos y que el dispositivo remoto los recibe:

Figura 59: Datos enviados por la placa WT12 y recibidos en el dispositivo remoto.

También, observamos como recibimos datos:

```
enviando info desde el portatil
```

Figura 60: Datos enviados por el dispositivo remoto y recibidos en la placa WT12.

Para cerrar la conexión, lo primero de todo es salir del modo de datos y pasar al modo comandos. Para ello, utilizamos la secuencia de escape *+++*. Ahora ya podemos cerrar la conexión indicando el id del enlace, en nuestro caso, *CLOSE 0*:

```
READY.
CLOSE 0
NO CARRIER 0 ERROR 0
```

Figura 61: Cierre de la conexión de manera correcta.

Como se observa en la Figura 61 y 62, el evento *READY* se produce cuando pasamos del modo de datos al modo comandos. Al introducir el comando *CLOSE 0*, la respuesta es *NO CARRIER 0 ERROR 0*. Lo que nos indica es, que se ha cerrado el enlace con identificador 0 sin errores.

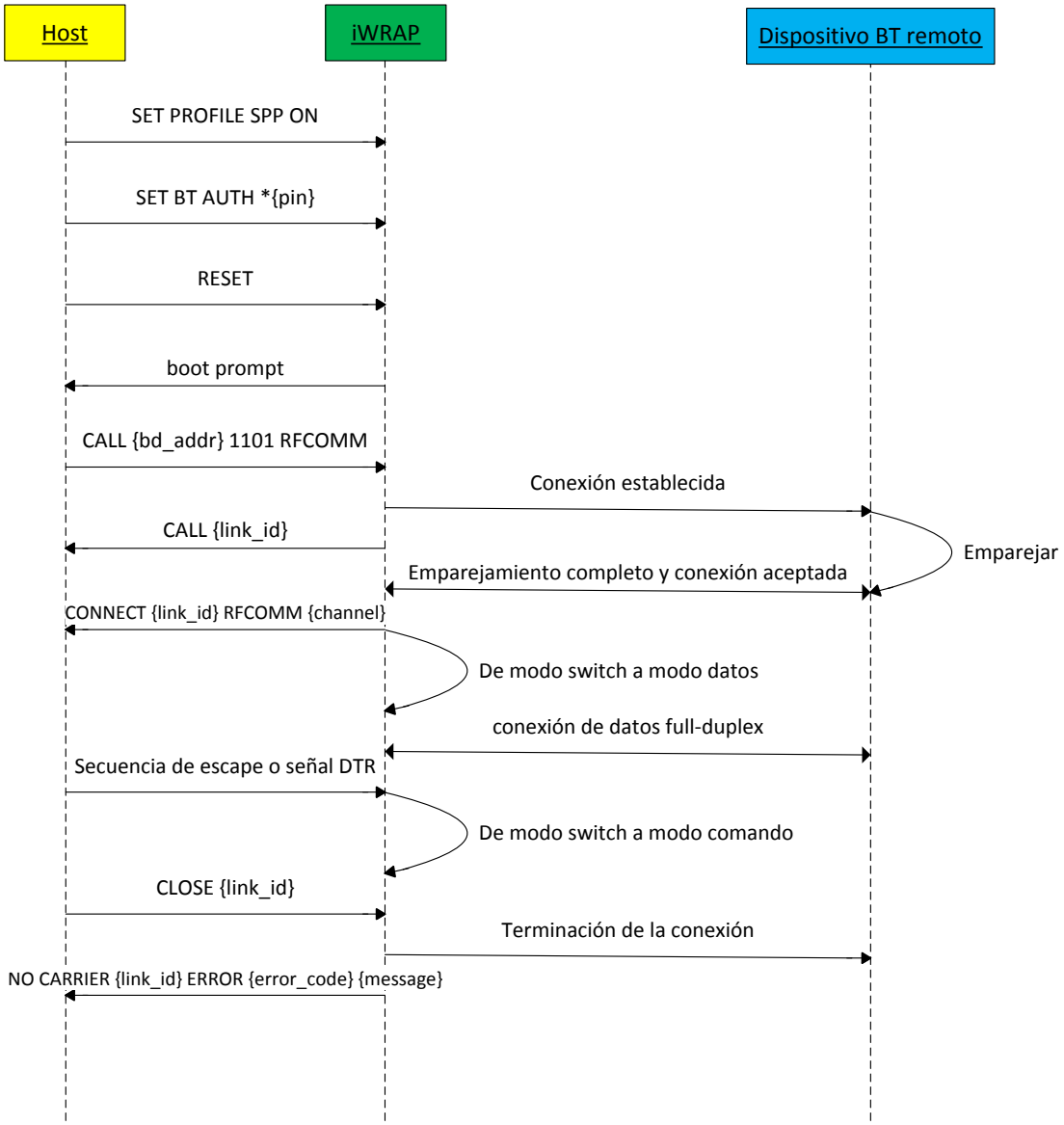


Figura 62: Diagrama de secuencia UML de una conexión SPP [iWRAP SPP 10].

3.6.2 Llamada entrante WT12 (Aceptador)

Ahora veamos el caso de cómo hay que configurar la placa WT12 para que tenga el rol de “aceptador” de la llamada Bluetooth (esclavo).

Como en el caso anterior, comprobaremos que la placa está encendida y funcionando correctamente con el comando *AT*. A continuación comprobaremos que está activado el perfil SPP con el comando *SET*. Si no lo está, lo activamos con *SET PROFILE SPP ON*. Siempre que se

haga algún cambio de configuración, se debe resetear el módulo Bluetooth para que los cambios realizados se hagan efectivos.

Una vez echa estas comprobaciones, activamos el puerto virtual serie del dispositivo Bluetooth que va a iniciar la conexión. Hacemos la llamada desde dicho dispositivo a la placa WT12.

Recibimos lo siguiente en el módulo WT12:

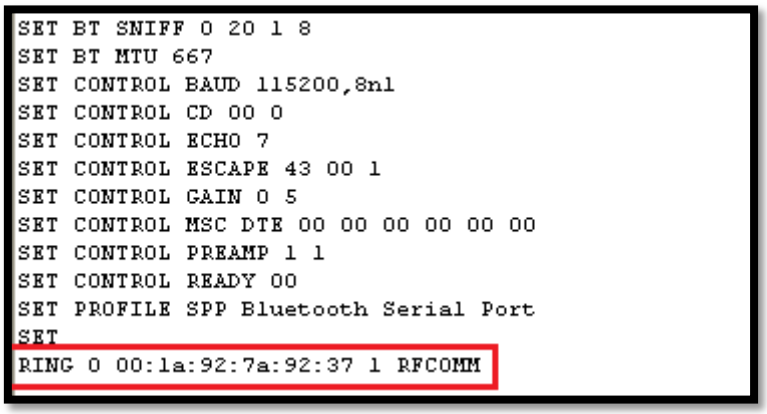


Figura 63: Llamada recibida desde el dispositivo maestro Bluetooth.

Esto significa que recibimos una llamada en el enlace 0, del dispositivo con dirección MAC Bluetooth que aparece en la figura y que la aceptamos en el canal 1 RFCOMM.

Comprobamos que podemos enviar y recibir datos:

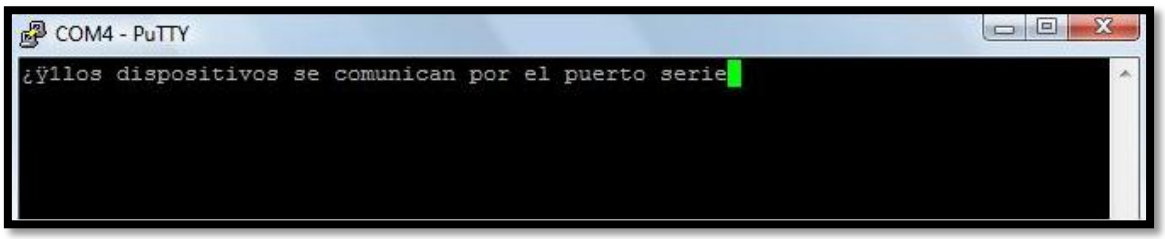


Figura 64: Recibiendo datos en el dispositivo maestro desde el esclavo.

Por último cerramos la conexión, primero saliendo del modo de datos como explicamos en el caso anterior.

Capítulo 4

Arquitectura y Especificación nSOM

4.1 INTRODUCCIÓN

El objetivo de este capítulo es describir la arquitectura y la especificación de un middleware basado en la computación orientada a servicios (SOC), el cuál proporciona un soporte para el desarrollo de de servicios simples distribuidos, lo que hace posible formar servicios más complejos a partir de ellos. Dentro del proyecto se ha utilizado el middleware nSOM y la ontología del lenguaje nSOL.

4.2 ARQUITECTURA ORIENTADA A SERVICIOS (nSOM)

Lo primero de todo, es definir el concepto de middleware. “Un middleware es un software que funciona como una capa de abstracción de software distribuida, que se sitúa entre las capas de aplicaciones y las capas inferiores. El middleware abstrae de la complejidad y heterogeneidad de las redes de comunicaciones subyacentes, así como de los sistemas operativos y lenguajes de programación, proporcionando una API para la fácil programación y manejo de aplicaciones distribuidas.

Dentro de las Redes Inalámbricas de Sensores, el middleware tiene que ser capaz de soportar la implementación de aplicaciones y tener en cuenta el funcionamiento básico de la red. Para ello, siempre hay que tener presente que lo nodos sensores están limitados por la autonomía de sus baterías. Otro aspecto fundamental es el rendimiento de la CPU, la memoria y el alcance de la interfaz radio. Existen diversos tipos de middleware, pero nos centraremos en el middleware orientado a servicios.

El paradigma de programación orientado a servicio (SOC) [Papazoglou&07] utiliza servicios para soportar el desarrollo de aplicaciones distribuidas que sean rápidas, de bajo coste, interoperables y escalables. Los servicios son autónomos, entidades independientes que pueden describir, ser publicados o ser descubiertos por el entorno en cuál estén presentes, y por supuestos intercambiar información mediante mecanismos de solicitud/respuesta. Como el resto de middlewares, el middleware SOC es independiente del lenguaje de programación en el que esté implementado y del sistema operativo donde se desarrolle.

Los Servicios Web son la tecnología SOC [Brazier&09] que más existo ha tenido. Pueden usar Internet como medio de comunicación o bien emplear protocolos de transporte y estándares abiertos, como por ejemplo el protocolo de acceso SOAP, como medio de

transmisión; el lenguaje de descripción de servicios WSDL, para publicación y definición de servicios, y para la orquestación de servicios el BPEL4WS.

La clave, de la programación orientada a servicios, es la arquitectura orientada a servicios (SOA). SOA es una manera lógica de diseñar un sistema software para proporcionar servicios tanto para aplicaciones de usuario final, como para otros servicios distribuidos en la red, a través de la publicación y descubrimiento de interfaces.

La propuesta de arquitectura middleware que se utiliza en el marco de nuestro proyecto [LifeWear Deliverable 12], es un middleware orientado a servicios llamado nSOM (nano Service_Oriented Middleware). Esta capa middleware se puede ver reflejada en la siguiente figura:

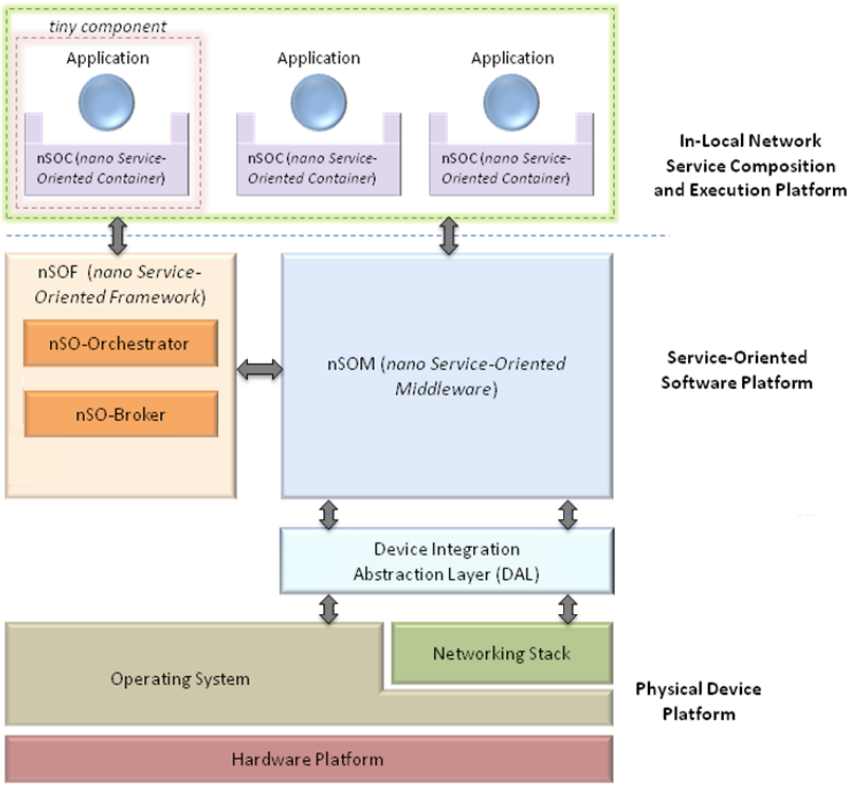


Figura 65: Arquitectura nSOM [LifeWear Deliverable 12].

La arquitectura nSOM está compuesta por tres capas de abstracción con respecto a las funcionalidades que cumplen cada una de ellas. Los niveles de abstracción son:

- *Plataforma del dispositivo físico inalámbrico (Wireless Physical Device Platform):* esta capa de abstracción de bajo nivel hace referencia al dispositivo llevable, incluyendo la plataforma hardware, el sistema operativo y a los detalles de la pila de red. Representa las características mínimas que debe tener el dispositivo a nivel hardware y software

para el desarrollo y ejecución del middleware, y sobre la cual estarán el resto de capas de abstracción.

- *Plataforma software orientada a servicio (Service-Oriented Software Platform):* esta capa de abstracción agrupa el middleware y el framework nSOF (nano Service-Oriented Framework), definiendo la capa software necesaria para abstraerse de la heterogeneidad de la red. Proporcionando un entorno integrado y distribuido para simplificar los problemas de programación, por medio de un conjunto de servicios genéricos, además de un punto de acceso a las funciones de orquestación e intermediación para los servicios semánticos.
- *Plataforma de ejecución y composición de servicios en red (In-Local Network Service Composition and Execution Platform):* representa la parte superior de nSOM. Esta capa está pensada para construir servicios a partir de pequeños componentes (bloques de construcción de servicio que están formados por un contenedor nSOC, nano Service-Oriented Container, y un componente software de aplicación.

Este middleware proporciona diferentes servicios, que se puede clasificar en diferentes grupos. Los grupos son:

- *Servicios de bajo nivel (Low level services):* estos servicios son obligatorios para el buen funcionamiento del middleware, y con ello el dispositivo llevable. Se usan intensivamente durante el uso normal de la arquitectura middleware. En la propuesta nSOM los servicios de bajo nivel son:
    - *Administración en tiempo real (Real-Time management):* este servicio proporciona soporte para la configuración de las capacidades en tiempo real del sistema operativo, que dependen de los requerimientos de los pequeños componentes.
    - *Comunicación (communication):* este servicio ofrece una abstracción del nivel de red. Configura una interfaz para abstraerse de las funcionalidades necesarias para la transmisión del mensaje, como la encapsulación o el enrutamiento.
    - *Administración del contexto de descubrimiento:* proporciona capacidades de descubrimiento y servicios de administración, los cuales se pueden usar en la capa de aplicación para buscar a los proveedores de servicio, así como sincronizar la lógica de negocio entre diferentes agentes de componentes desplegados en diferentes dispositivos llevables.

- *Servicios de alto nivel (High level services):* esta capa presenta servicios de abstracción de alto nivel. Desde un punto de vista completo del sistema, estos servicios son accesibles por una aplicación llamada Application Tiny Components lo que mejora el nodo llevable y la red con un valor añadido, haciendo que el middleware se adapte mejor a diferentes escenarios, y proporcionando soporte a la plataforma de despliegue y composición de servicios. Los servicios de alto nivel son:
    - *Consulta (Query):* este servicio permite al middleware gestionar mensajes de consulta, tanto para las comunicaciones intra-nodo e inter-nodo, como para solicitar información de interés.
    - *Configuración de servicio In-node (In-node service configuration):* este servicio está enfocado a proporcionar al middleware servicios con soporte de configuración (instalación hardware, configuración de contexto de adquisición, políticas de seguridad, configuración de enrutamiento, cuestiones de gestión del conocimiento y del razonamiento, funciones de agregación, etc.).
    - *Comando (command):* ofrece al middleware la posibilidad de crear y añadir comandos, para gestionar las pequeñas aplicaciones en la composición del servicio y desplegar la plataforma.
- *Servicios multinivel (Cross-layer services):* este tipo de servicios proporciona funcionalidades ortogonales, que están relacionados con funciones de alto y bajo nivel, tanto para el dispositivo llevable como para la red. Los servicios multinivel dan soporte a tareas desarrolladas en los servicios de alto y bajo nivel. Se distinguen los siguientes servicios multinivel:
    - *Entorno de ejecución de código llevable (Portable Code Execution Environment):* este servicio soporta el despliegue y la ejecución de código llevable desde otros dispositivos llevables, siguiendo un paradigma de código móvil.
    - *Razonamiento (reasoning):* este servicio ofrece soporte para modificar de manera dinámica no solo el comportamiento del dispositivo llevable, sino también los servicios desplegados en él, de acuerdo a un conjunto de reglas y utilizando para ello un motor de inferencia. El motor “Reasoning Engine” será el “cerebro” de la arquitectura nSOM, la parte que proporcionará conocimiento del contexto global al sistema de acuerdo a una filosofía de distribución.
    - *Seguridad (security):* define el mecanismo de seguridad para la provisión de servicios seguros de confidencialidad, integración y autenticación, añadiendo

también protección al dispositivo llevable. La especificación de seguridad se define en dos partes. Por una parte, es necesario definir los servicios seguros ofrecidos en la red llevable como: la autenticación, la integridad y la confidencialidad. Por otra, se deben definir mecanismos de seguridad que ayuden a proteger la red de posibles ataques que puedan ocurrir, intentando que no se produzcan, y si así fuera que el daño que se le cause a la red sea mínimo.

- *Servicios de control (Control services):* representan el núcleo del middleware. Proporcionan el despliegue de componentes y la administración del ciclo de vida, así como la comunicación inter-componente a través del uso de eventos. Este diseño de la capa de control libera a componentes de otras capas de la carga computacional, así ellos solo tienen que lanzar eventos a esta capa sin que empeore su rendimiento. Los servicios de control los proporcionan dos entidades funcionales:
    - *nContainer:* esta entidad administra el ciclo de vida (instanciar, cargar, iniciar, ejecutar, detener y destruir) de los componentes del middleware, ofreciendo capacidades de scheduling y de dispatching.
    - *Publish/Suscriber Kernel:* ofrece capacidades para publicar (anuncia la generación de un evento) y de suscripción (anuncia el consumo de un evento) a los componentes middleware y a los componentes pequeños, junto con la gestión de eventos y la función de dispatching.

Todos los servicios y el modelo de componentes se pueden ver en la siguiente Figura 66:

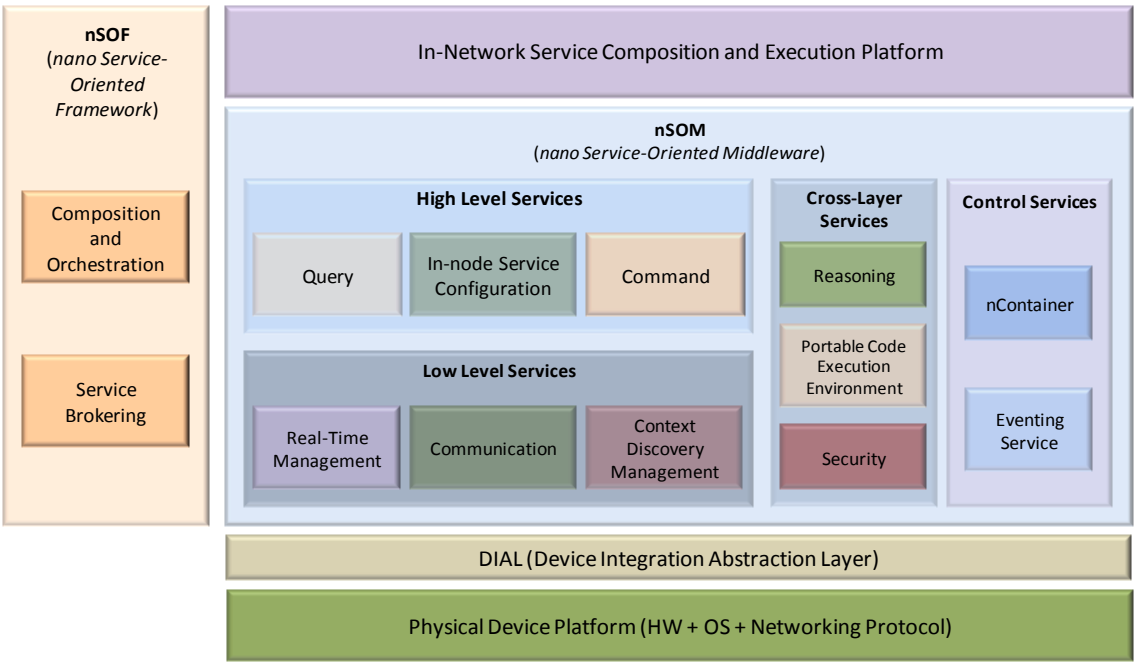


Figura 66: Aproximación del modelo de componentes y de los servicios en nSOM [LifeWear Deliverable 12].

4.3 AGENTES

Lo primero de todo es definir el concepto de agente. Un agente software [Brazier&09] es una "*entidad software que funciona de forma autónoma y continua en un entorno concreto*".

En nuestro caso, se encargan de realizar las medidas relacionadas con los servicios que tanto el usuario como la red soliciten, como por ejemplo la temperatura ambiente de una habitación o los parámetros fisiológicos como el ritmo cardiaco, la temperatura corporal y/o la tasa respiratoria del usuario. Los diferentes agentes tienen una interfaz común que consta de varios métodos, los cuales se definirán a continuación:

MÉTODOS	FUNCIÓN
public void init()	Inicia el agente.
public void load()	Carga el agente.
public void run()	Comienza el funcionamiento del agente
public void stop()	Para el agente.
public void destroy()	Desconecta el agente.
public void unload()	Descarga el agente.
public void receptacle (ServiceContext operationInvocation)	Implementa el receptáculo del agente. Se invoca para pasar las invocaciones del

	servicio a los agentes.
Public nSOMServiceAgentObject getServiceAgentObject()	Obtiene la descripción del servicio para este agente.
Public String getnSOMAgentDescription()	Proporciona el nombre del agente que ha sido invocada.

Tabla 9: Interfaz general de un agente.

En nuestro caso, existe un agente llamado nSOMZephyrAgent que se encarga de implementar un agente que proporciona servicios correspondientes al dispositivo Zephyr Bioharness explicado en capítulos anteriores. En concreto, proporciona 3 servicios obtenidos del módulo Zephyr y otro servicio de temperatura ambiente que recoge de un nodo sensor de la red de la arquitectura. Los métodos de la interfaz de dicho agente se describen a continuación:

MÉTODOS	FUNCIÓN
public nSOMZephyrAgent()	Constructor de la clase nSOMZephyrAgent.
public int getHeartRate()	Mide el ritmo cardiaco.
public double getBreathingRate()	Mide la tasa respiratoria.
public double getBodyTemperature()	Mide la temperatura corporal.
public void catchTemperature(int temp)	Mide la temperature ambiente.

Tabla 10: Interfaz del agente nSOMZephyrAgent.

Los agentes pueden proporcionar servicios simples ys ervicios compuestos. En el caso del agente nSOMZephyrAgent solo proporciona servicios simples. La publicación de los servicios se hace utilizando el formato JSON.

4.4 JSON, SMD, RDF Y NSOL

4.4.1 JSON

JSON (JavaScript Object Notation - Notación de Objetos de JavaScript) es un formato ligero de intercambio de datos. Leerlo y escribirlo es simple para humanos, mientras que para las máquinas es simple interpretarlo y generarlo. Está basado en un subconjunto del Lenguaje de Programación JavaScript , Standard ECMA-262 3rd Edition - Diciembre 1999 . JSON es un formato de texto que es completamente independiente del lenguaje pero utiliza convenciones que son ampliamente conocidos por los programadores de la familia de lenguajes C, incluyendo C, C++, C#, Java, JavaScript, Perl, Python, y muchos otros. Estas propiedades hacen que JSON sea un lenguaje ideal para el intercambio de datos.”

JSON está formado por dos estructuras:

- Una colección de pares de nombre/valor. Lo que se conoce como un *objeto*, registro, estructura, diccionario, tabla hash, lista de claves o un arreglo asociativo.
- Una lista ordenada de valores. En la mayoría de los lenguajes, esto se implementa como arreglos, vectores, listas o secuencias.

4.4.2 SMD

Un servicio SMD (Service Mapping Description) es una representación JSON que describe servicios web, puede definir varios aspectos de un servicio web para que los clientes puedan interactuar de manera coherente con estos servicios web. Un SMD puede ser usado por herramientas genéricas para generar interfaces que accedan a los servicios web disponibles. El formato SMD ha sido diseñado para ser flexible, simple, compacto, legible y fácil de implementar.

Un SMD se representa como un objeto JSON con atributos que se usan para definir la interacción entre los servicios web disponibles. Un SMD puede tener atributos relacionados con los servicios que se ofrecen, métodos relacionados con esos atributos y elementos que se corresponden con los servicios disponibles, así como nuevos servicios. A su vez, cada uno de los elementos correspondientes a esos servicios puede tener su propio conjunto de atributos. Los atributos definidos en el servicio raíz son heredados por aquellos servicios que no posean atributos definidos.

4.4.3 RDF

El marco de descripción de recursos (Resource Description Framework, RDF) es un framework para metadatos en la Worl Wide Web, desarrollado por W3C [Decker&00]. Es un lenguaje de objetivo general para representar la información en la web. Este modelo se basa en la idea de convertir las declaraciones de los recursos en expresiones con la forma sujeto-predicado –objeto (conocidos como tripletes). El sujeto es el recurso, es decir aquello que se está describiendo. El predicado es la propiedad o relación que se desea establecer a cerca del recurso. Por último, el objeto es el valor de la propiedad o el otro recuso con el que se establece la relación.

4.4.4 nSOL

Para el servicio de descripción, descubrimiento, e invocaciones se utiliza un lenguaje ligero llamado nSOL (nano Semantic Ontology Language), usando SMD sobre JSON con RDF (Notación 3) para obtener una herramienta ligera y potente para el desarrollo de Ecosistemas de Inteligencia Ambiental.

La anotación semántica seguirá las reglas definidas en las ontologías desarrolladas ad-hoc, proporcionando la composición de nuevos servicios que serán anotados semánticamente.

send: { "transport": "j2me.radiogram", "envelope": "JSON-2.0", "target": "
0014.4F01.0000.752E/nSOMZephyrAgent", "origin": "
0014.4F01.0000.79E3/nSOMBrokerAgent", { "operation": "getBodyTemperature", "
parameters": [void] , "result": "877737.2" } }

Figura 67: Ejemplo de publicación del servicio temperatura corporal en formato JSON.

4.5 SUBSITEMAS DE LA ARQUITECTURA NSOM

En la arquitectura global se pueden distinguir varios subsistemas, que son: subsistema orientado a servicios, subsistema ESB, subsistema de nodos con servicios de temperatura y subsistema de nodos Bluetooth. A continuación, se explicará cada uno de dichos subsistemas.

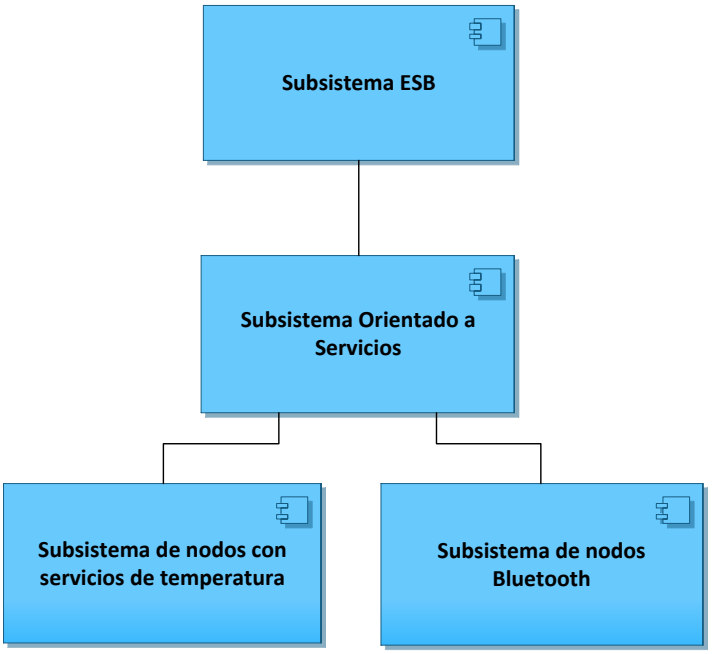


Figura 68: Subsistemas de la arquitectura nSOM.

4.5.1 Subsistema orientado a servicios

Dentro de la arquitectura del sistema orientado a servicios hay un subsistema muy importante que interactúa con el resto de dispositivos. Este subsistema está formado por dos componentes software: el Broker y el Orquestador.

4.5.1.1 Broker

Este componente se encarga de la exposición y solicitud de servicios. Si se trata de un servicio simple, no se lo comunicará al orquestador y lo publicará hacía fuera. Por el contrario, si se trata de un servicio compuesto, el Broker se lo tendrá que comunicar al Orquestador, ya que es el encargado de componer servicios más complejos a partir de servicios simples. Una vez obtenido el servicio compuesto, el Broker se encargará de publicarlo.

En la práctica, este componente está implementado en una mota SunSpot.

4.5.1.2 Orquestador

Es el componente encargado de interactuar con el Broker para componer servicios a partir de servicios simples. También interactúa con los agentes en caso de que sea necesario para realizar un servicio compuesto.

Cuando se solicita la composición de un servicio existe un tiempo limitado para su construcción. Si vence este temporizador antes de recibir todos los datos necesarios, el Orquestador considerará que el servicio compuesto no está disponible y se lo comunicará al Broker para que se lo indique al usuario final.

Al igual que el Broker, en la práctica este componente está implementado en una mota SunSpot.

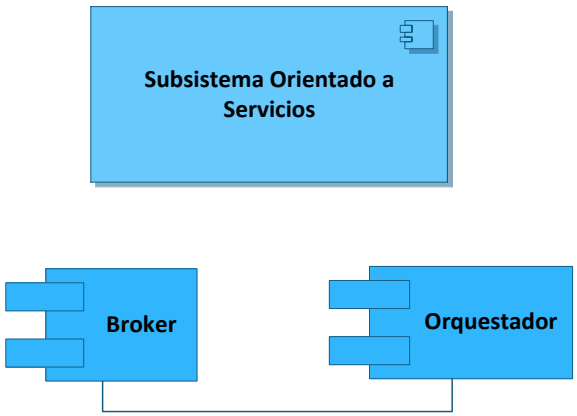


Figura 69: Subsistema orientado a servicios, formado por dos componentes (Broker y Orquestador).

4.5.2 Subsistema ESB

Otro elemento importante de nuestra arquitectura de red es el ESB [LifeWear ESB 12]. El ESB, del inglés Enterprise Service Bus, es la columna vertebral de una implementación SOA ofreciendo una capa de abstracción en la parte superior de una implementación de un sistema de mensajes de empresa, que permite a los expertos en integración explotar el valor del envío de mensajes sin tener que escribir código. Un bus de servicio empresarial se construye a partir de funciones base que se dividen en sus partes constituyentes, con una implantación distribuida cuando sea necesario, de modo que trabajen bajo demanda. Con esta arquitectura software se resuelve el problema de la heterogeneidad entre diferentes aplicaciones, plataformas y procesos.

Un estándar ESB proporciona diversos servicios, entre ellos, transformación dinámica de mensajes, seguridad, enrutamiento inteligente.

En nuestro proyecto, necesitábamos encontrar una solución para la integración de diferentes middleware. De ahí la creación del ESB Lifewear. Siguiendo la especificación de la

arquitectura general cliente-servidor, las solicitudes del cliente son encaminadas (y adaptadas si fuera necesario) al correspondiente servicio, que responderá a la petición. De hecho, el cliente envía la solicitud al ESB (en vez de enviar a la aplicación del servidor), y es el ESB el responsable de encaminar la solicitud al servidor, y de monitorizar y registrar todo el tráfico. Esta propuesta, dispone de una única interfaz externa para acceder a cualquiera de las aplicaciones que se encuentran por detrás del ESB. Lo que hace que los problemas de actualización y migración sean más fáciles de resolver.

En una arquitectura empresarial que hace uso de un ESB, una aplicación se comunicará a través del bus, el cual actúa como un único mensaje intercambiable entre las aplicaciones. Esta propuesta reduce el número de conexiones punto a punto entre aplicaciones que se comunican entre sí. Esto, a su vez, hace que el análisis del impacto para los cambios importantes de software sea más simple y más directo. El bus también se puede distribuir, adquiriendo las ventajas de escalabilidad, flexibilidad, etc.

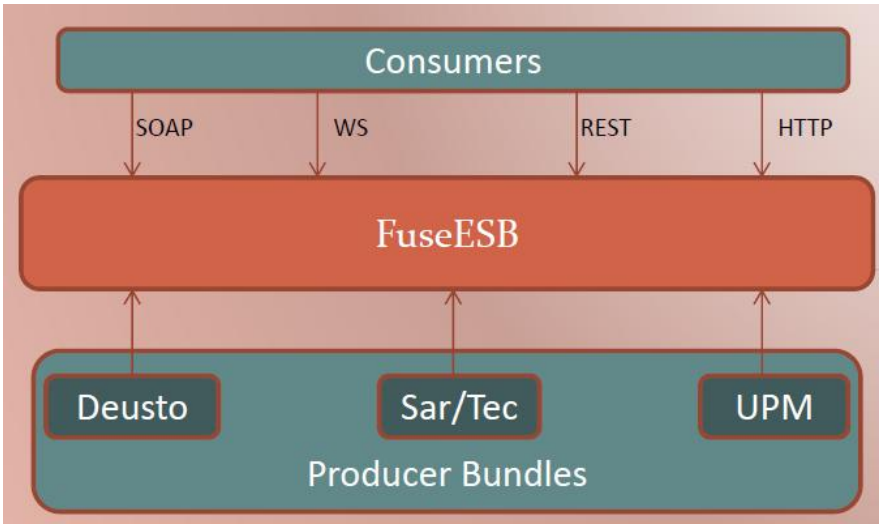


Figura 70: Visión de conjunto del ESB Lifewear [LifeWear ESB 12].

En el ESB Lifewear se han definido varia interfaces para que los clientes accedan a los servicios publicados en el ESB. Para simplificar y aligerar a los clientes, se han usado el protocolo REST y mensajes JSON.

REST, en inglés Representation Stale Transfer, es un estilo de arquitectura distribuida para sistemas hipermedia distribuidos, inicialmente descrito en el contexto HTTP, pero que no está limitado a dicho protocolo. Las arquitecturas RESTful (conforme a las limitaciones REST) se basan en el protocolo HTTP, pero se pueden usar en otros protocolos de la capa de aplicación, siempre que proporcionen una transferencia representacional del estado. Básicamente, es una

arquitectura cliente/servidor donde los clientes envían peticiones a los servidores. Los servidores reciben y procesan la solicitud y devuelven la respuesta. La unidad básica de información es el recurso: un objeto único que puede ser localizado a través de una URL.

El ESB Lifewear reside en un PC, con sistema operativo Ubuntu, independiente al que se puede acceder a través de un IP pública. Conectado a él tiene una estación base SunSpot que hace de sumidero (Gateway) con la red inalámbrica de sensores.

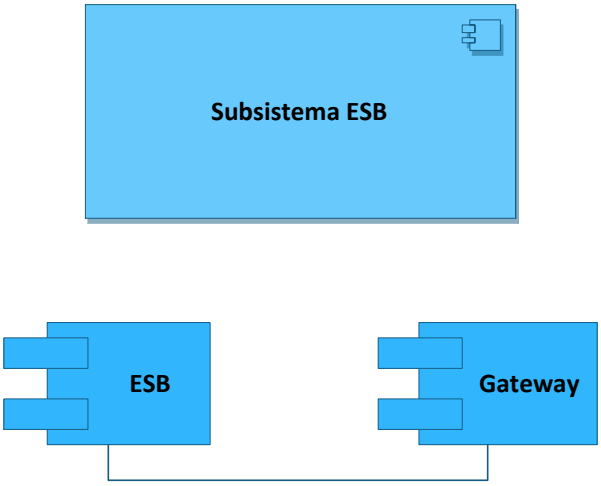


Figura 71: Subsistema ESB formado por los componentes ESB y Gateway.

4.5.3 Subsistema de nodos con servicios de temperatura

Dentro de la arquitectura de red, a parte del Orquestador, del Broker y del ESB (más estación base) existen otros nodos SunSpot. Estos son los nodos que implementan agentes que ofrecen servicios simples de temperatura ambiente.

En total, son 3 motas SunSpot las que ofrecen un servicio de temperatura. Denominados: temperatura1, temperatura2 y temperatura3. Estos nodos alojan, cada uno, un agente que proporciona la temperatura ambiente de la zona en la que se encuentran.

Posteriormente, a partir de estas mediciones, el Orquestador es capaz de componer un servicio complejo, siempre a través del Orquestador como explicamos en apartados anteriores.

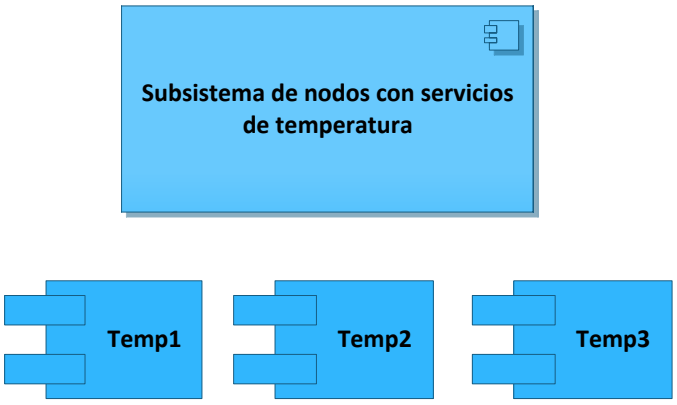


Figura 72: Subsistema de nodos de servicios de temperatura, compuesto por temp1, temp2 y temp3.

4.5.4 Subsistema de nodos Bluetooth

También hay dos nodos SunSpot [SunSpot eDemo 10] al que se les ha instalado dos placas Bluetooth. Ambos forman un subsistema de nodos con integración de una interfaz Bluetooth.

4.5.4.1 Instalación Hardware

En el último modelo de la mota SunSpot (revisión 8) se ha incluido una nueva placa, llamada eDemoBoard, que incorpora una nueva USART (P2, en la figura).

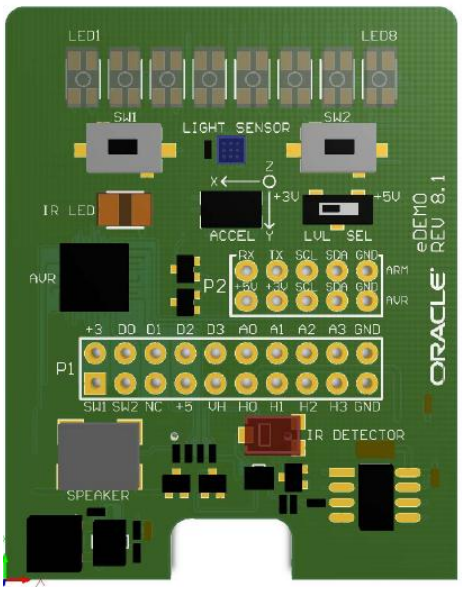


Figura 73: eDemoBoard de la SunSpot [SunSpot eDemo 10].

Los pines que se han utilizado, de la zona P2, se muestran en la siguiente tabla con sus correspondientes descripciones:

Número PIN	Nombre señal	Propiedad	Descripción
2	RXD	Entrada	ARM USART de 3V ó 5V (de serie) RXD0. Nivel seleccionado por el interruptor SW3.
4	TXD	Salida	ARM USART de 3V ó 5V (de serie) TXD0. Nivel seleccionado por el interruptor SW3.
10	GND	Potencia	Toma de tierra.

De la parte P1, se ha utilizado un único pin:

2	VCC	Potencia	+3V de potencia de la placa principal SPOT.

Tabla 11: Descripción de los pines que se han utilizado de la SunSpot.

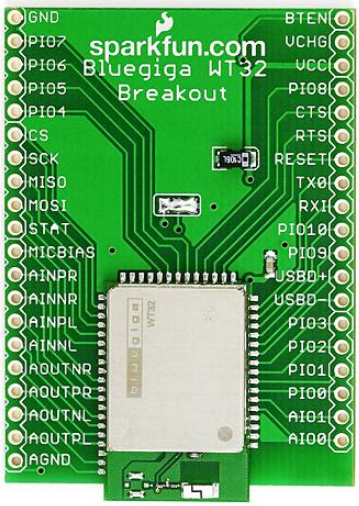


Figura 74: Placa Bluetooth Sparkfun Bluegiga WT32 [Sparkfun 12].

Estos 4 pines han sido soldados a los pines correspondientes de la placa: sparkfun.com Bluegiga WT32 Breakout. En concreto, se han unido de la siguiente manera:

PIN EDEMO BOARD SUNSPOT	PIN PLACA WT32 SPARKFUN
RXD	TXO
TXD	RXI
GND	GND
VCC	VCC

Tabla 12: Pines soldados entre la SunSpot y la placa Sparkfun WT32.

En las siguiente Figura 75 se puede ver el estado final de la mota SunSpot con la placa Bluetooth WT32 incluida:

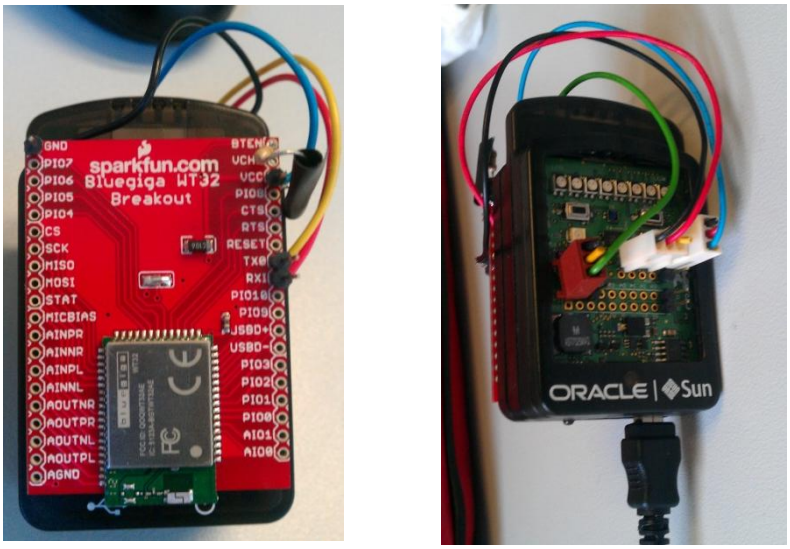


Figura 75: Pines soldados en la placa Sparkfun (izq.) y en la SunSpot (dcha.).

4.5.4.2 Configuración Software

Una de las motas SunSpot aloja el agente nSOMZephyrAgent, mencionado anteriormente. Este nodo se encarga de recoger los datos que provienen del módulo Zephyr Bioharness a través de un driver Bluetooth (ANEXO I). A su vez, esta mota es la encargada de comunicarse, a través de la interfaz radio IEEE 802.15.4, con el resto de componentes que forman la red. En la Figura 76 esta mota equivale al componente ZephyrBT. Además dispone de un servicio de alarmas (ANEXO II):

- Alarma por baja temperatura ambiente: se produce esta alarma cuando la temperatura ambiente está por debajo de 12 ºC.
- Alarma por alta temperatura ambiente: esta alarma se activa cuando la temperatura ambiente está por encima de 34º C.

- Alarma por baja temperatura corporal: se produce esta alarma cuando la temperatura ambiente está por debajo de 36,4 ºC.
- Alarma por alta temperatura corporal: si la temperatura corporal del usuario supera el umbral de 37,6 ºC se activará dicha alarma.
- Alarma por bajo ritmo cardiaco: si el ritmo cardiaco del usuario está por debajo de 50 pulsaciones por minuto se activará una alarma.
- Alarma por alto ritmo cardiaco: esta alarma se activa cuando las pulsaciones de la persona son superiores a 120 ppm.

Una vez que se ha producido una alarma, esta se pasa al otro nodo que dispone de interfaz Bluetooth, usando una comunicación radio Unicast (IEEE802.15.4). Este nodo, es el otro componente del subsistema (AlarmasBT), que simplemente ejerce como pasarela entre el componente ZephyrBT y el reloj WiMM que se describió en apartados anteriores.

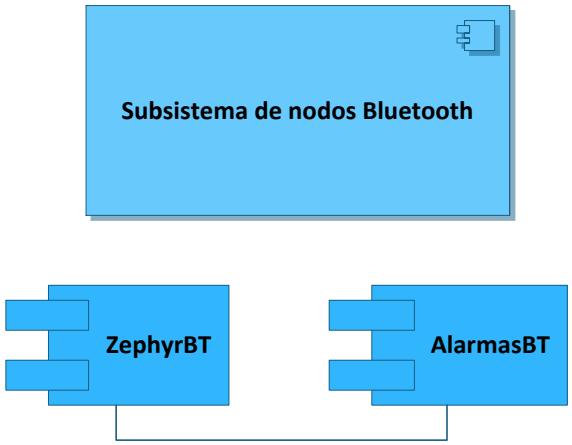


Figura 76: Subsistema de nodos Bluetooth, formado por los componentes ZephyrBT y AlarmasBT.

4.6 MODELO DE COMUNICACIÓN

Como ya se ha comentado en apartados anteriores, nuestra arquitectura orientada a servicios nSOM, se ocupa de la publicación y solicitud de servicios. A continuación, vamos a ver un par de ejemplos, donde se mostrarán la exposición de servicios simples y servicios compuestos.

4.6.1 Ejemplos de servicios simples

En primer lugar, vamos a ver un ejemplo, a través de un diagrama UML de secuencia, de cómo el agente nSOMZephyrAgent publica sus servicios simples hacia el Broker. Como se puede observar en la Figura 77, el Orquestador no interviene para nada. Ya que como mencionamos en apartados anteriores, únicamente se encarga de componer servicios.

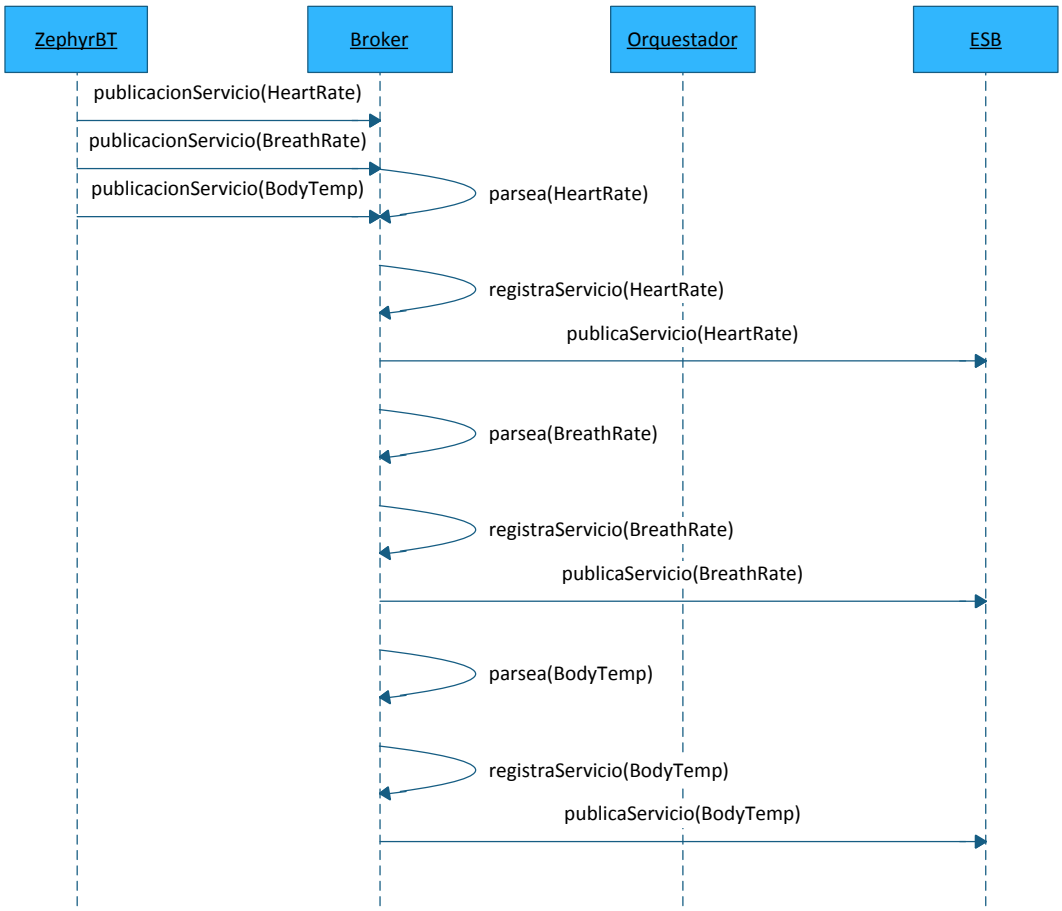


Figura 77: Diagrama UML de secuencia para la exposición de servicios simples.

Nuestro Agente, también necesita el parámetro de temperatura ambiente para el servicio de alarmas. Este parámetro nos lo facilita la mota Temp2, que vimos anteriormente en el subsistema de nodos con servicios de temperatura. Veamos, en la Figura 78, como nuestro agente consigue dicho parámetro a través del Broker, siendo el resto de la red transparente para él.

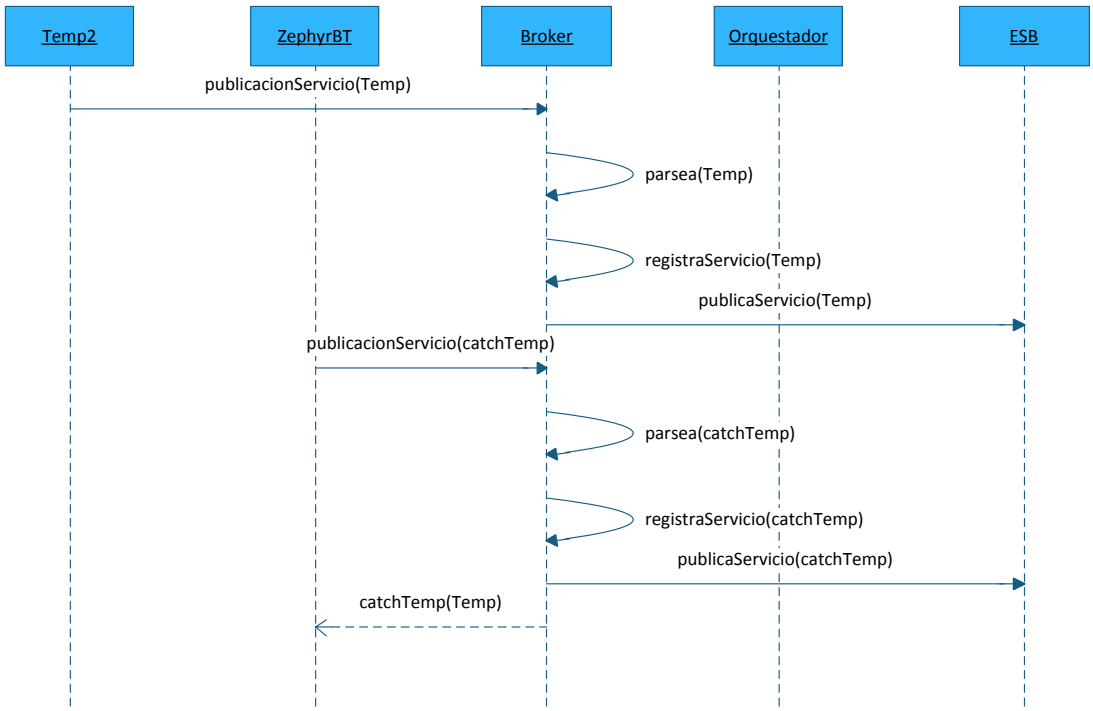


Figura 78: Diagrama de secuencia UML para la publicación de un servicio simple, que a su vez, es empleado por otra mota de la red.

4.6.2 Ejemplo de servicio compuesto

En este apartado vamos a ver como el Orquestador es capaz de componer un servicio a través de varios servicios simples. En concreto, vamos a realizar la composición del servicio compuesto “Injury Prevention” que se emplea para prevenir una lesión muscular. A partir de los servicios simples temperatura ambiente, temperatura corporal y ritmo cardiaco, podemos conocer si una persona tiene un bajo, medio o alto riesgo de padecer una lesión. Veamos en la siguiente figura como se compone dicho servicio:

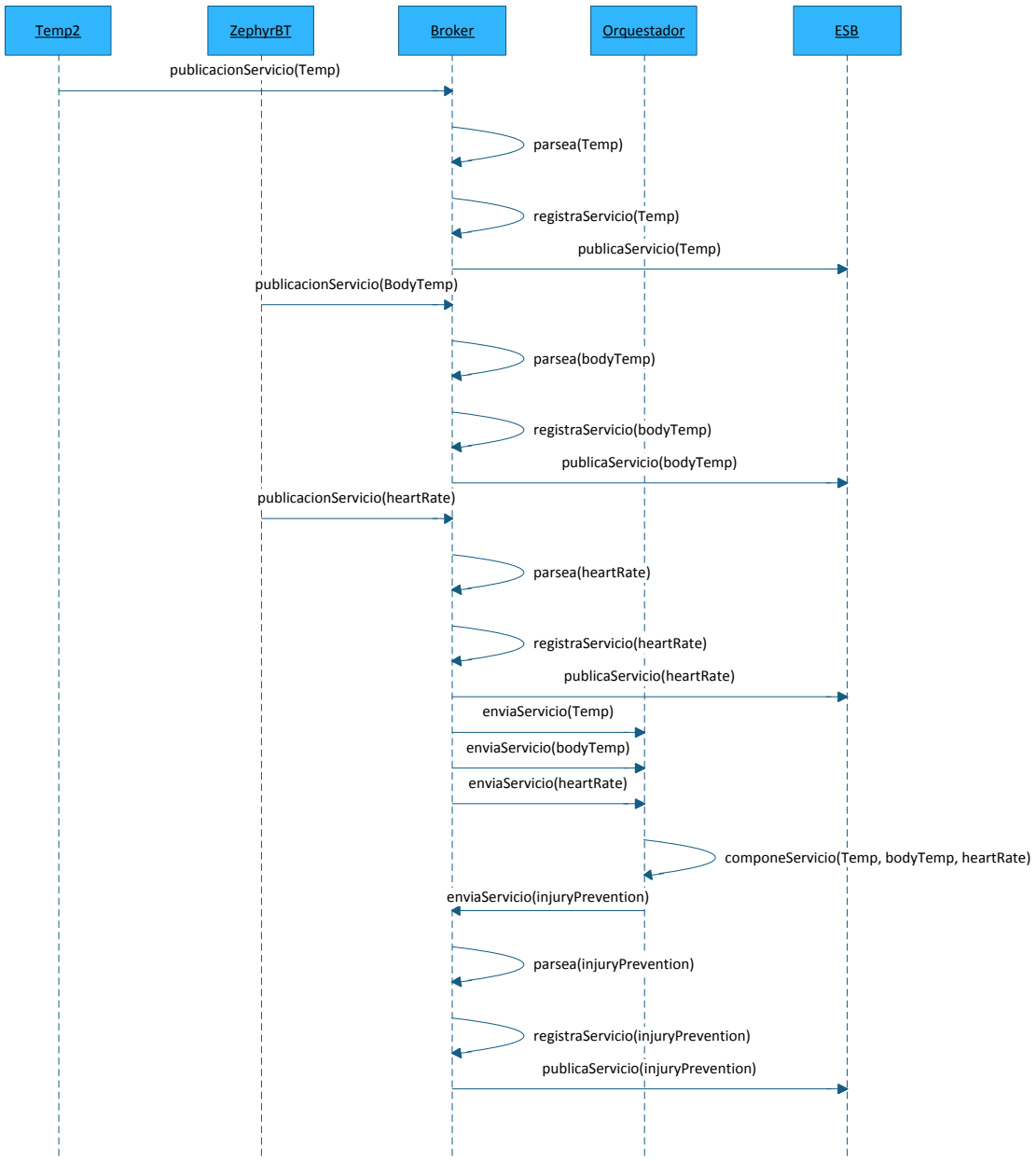


Figura 79: Diagrama de secuencia UML para la composición del servicio "Injury Prevention".

4.7 DISEÑO DETALLADO DE LA ARQUITECTURA

A continuación vamos a ver el diseño detallado de los dos nodos Bluetooth. Primero veremos, a través de un diagrama de clases, el diseño del nodo que hemos denominado nodo ZephyrBT, que como mencionamos anteriormente se encarga de tomar datos del módulo Bioharness Zephyr y facilitárselo a la red de sensores inalámbricos. Además, tiene un servicio

de alarmas que en caso de saltar una alarma se lo transmite, vía radio, a la otra mota AlarmasBT.

Después, describiremos la arquitectura detallada del nodo AlarmasBT, que es el encargado de hacer de pasarela entre el nodo ZephyrBT y el reloj WiMM para pasarle a este una alarma en tiempo real.

4.7.1 Diseño detallado de la arquitectura del nodo ZephyrBT

En la Figura 80, podemos observar el diagrama UML de clases más representativas del nodo Zephyr BT. Este nodo dispone de otras muchas clases que emplea la arquitectura nSOM.

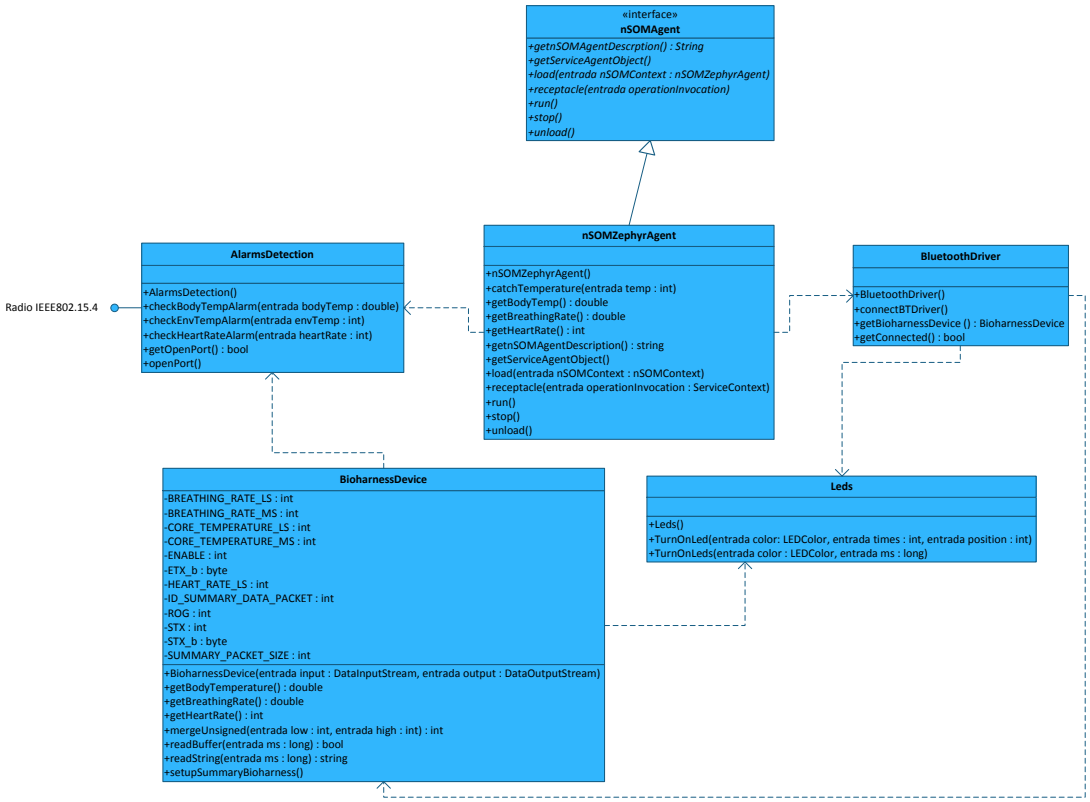


Figura 80: Diagrama de clases UML para el nodo ZephyrBT.

La clase nSOMZephyrAgent hace uso de la interfaz nSOMAgent, al igual que el resto de agentes de la arquitectura nSOM. Dicha clase, se encarga de todo lo que respecta al agente y depende de las clases AlarmsDetection y BluetoothDriver. La clase AlarmsDetection se emplea para la detección de las alarmas, y si la hay, enviarla a la otra mota SunSpot a través de la interfaz radio IEEE802.15.4.

Por otro lado, la clase BluetoothDriver depende de la clase Leds y de la clase BioharnessDevice. La clase Leds se encarga de iluminar los leds de la mota SunSpot. La clase BluetoothDriver es la encargada de establecer la comunicación Bluetooth entre la mota y el módulo Bioharness, y a su vez, depende de las clases Leds y BioharnessDevice.

Por último la clase BioharnessDevice, es el parser de los datos que se envían desde el cinturón Zephyr Bioharness a la mota. Es decir, es la encargada de la lectura de datos: ritmo cardiaco, tasa respiratoria y temperatura corporal de la persona.

En el Anexo I, tenemos el listado completo de cada clase representada en la Figura 80, con sus métodos y atributos, y una breve descripción de ellos.

4.7.2 Diseño detallado de la arquitectura del nodo AlarmasBT

En la siguiente figura se puede apreciar, mediante un diseño UML de clases, el funcionamiento de la mota AlarmasBT.

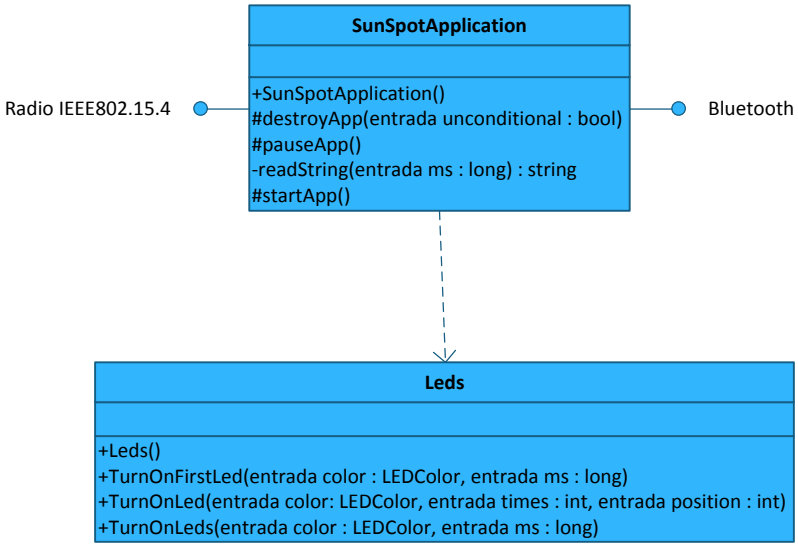


Figura 81: Diagrama de clases UML para el nodo AlarmasBT.

Este nodo, solo implementa dos clases. La clase SunSpotApplication, es la clase encargada de ejercer de pasarela entre la mota ZephyrAgent y el reloj WiMM. Recibe las alarmas a través de la interfaz radio IEEE802.15.4 y se las transmite, en tiempo real, al reloj WiMM por la interfaz Bluetooth.

La clase Leds, es una clase auxiliar que se encarga de encender los leds de la mota SunSpot cuando se recibe una alarma.

Este diagrama se completa con la API que aparece en el Anexo II, donde se podrá entender mejor que hace cada clase, con la descripción de sus métodos y atributos.

Capítulo 5

Validación del sistema y análisis de resultados

5.1 INTRODUCCIÓN

En apartados anteriores, se han tratado las fases de diseño e implementación de la arquitectura orientada a servicios nSOM del proyecto Lifewear. En este capítulo, se van a tratar la validación del software desarrollado y la verificación de los resultados obtenidos.

Para llevar a cabo la fase de validación del sistema, en primer lugar analizaremos el escenario de la aplicación real donde se ha desplegado la red y se han realizado todas las pruebas y mediciones. Una vez realizado el despliegue, y con el sistema ya funcionando, se realizarán una serie de pruebas sobre la Infraestructura LifeWear, centrándonos básicamente en los dispositivos Bluetooth. Posteriormente, cuando dispongamos de los resultados, se analizarán.

5.2 JUSTIFICACIÓN DEL ESCENARIO PLANTEADO

La validación de la Infraestructura LifeWear que hemos especificado y, posteriormente implementado, se lleva a cabo en un escenario tal como el gimnasio de la Universidad. Este escenario del deportista se utilizará tanto para demostrar el correcto funcionamiento de la red como para ver visualmente la disposición de todos los nodos y dispositivos de la red.

Se medirán parámetros relacionados con las interfaces Bluetooth que aparecen en la Infraestructura orientada a servicios, por ejemplo lo que tarda en conectarse/desconectarse la mota ZephyrBT al módulo Zephyr Bioharness. O el tiempo que tarda el agente de dicha mota en enviar el mensaje "*Hello*" a la red para registrar todos sus servicios.

El escenario planteado estará formado por una serie de sensores de temperatura, en concreto, Temperatura1, Temperatura2 y Temperatura3. Un nodo con la función de Broker, otro nodo con la función de Orquestador. Además de un PC que será el ESB de la arquitectura al que estará conectada una estación base que hará la función de Gateway. El deportista llevará consigo el cinturón Bioharness Zephyr, el reloj WiMM y un cinturón con las dos motas SunSpot que disponen de interfaz Bluetooth, que son ZephyrBT y AlarmasBT.

5.3 ESTUDIO DE LOS CASOS DE USO DEL ESCENARIO

Dentro del escenario planteado, vamos a analizar la parte Bluetooth. En concreto, vamos a analizar los casos de uso de la mota ZephyrBT y de la mota AlarmasBT.

5.3.1 Infraestructura de la mota ZephyrBT

En la infraestructura de la mota ZephyrBT se distingues seis casos de uso: Conectar BT, exponer servicios, pedir servicios, recibir datos del Zephyr, enviar alarmas y desconectar BT. En la Figura 82 se pueden ver reflejados los casos de uso.

5.3.1.1 Actores

Dentro de esta infraestructura se han definido 5 actores que pueden interactuar con nuestra mota:

- *Agente*: se corresponde con el agente nSOMZephyrAgent, que se mencionó en apartados anteriores y que está desplegado en la mota ZephyrBT.
- *Broker*: desempeña las funciones de Broker que se describieron en el apartado 4.5.5.1. Está desplegado en un nodo de la red de sensores inalámbricos.
- *Cinturón Zephyr*: representa el módulo Bioharness Zephyr definido en el apartado 2.2.8.3. Se encarga de enviar datos a la mota ZephyrBT, una vez establecida una conexión Bluetooth.
- *Usuario*: persona que puede acceder a la aplicación, a través del ESB, para obtener servicios, tanto simples como compuestos.
- *Mota Alarmas BT*: representa a la otra mota SunSpot que lleva el deportista. Cuando salta una alarma, la mota Zephyr BT se la envía a dicha mota.

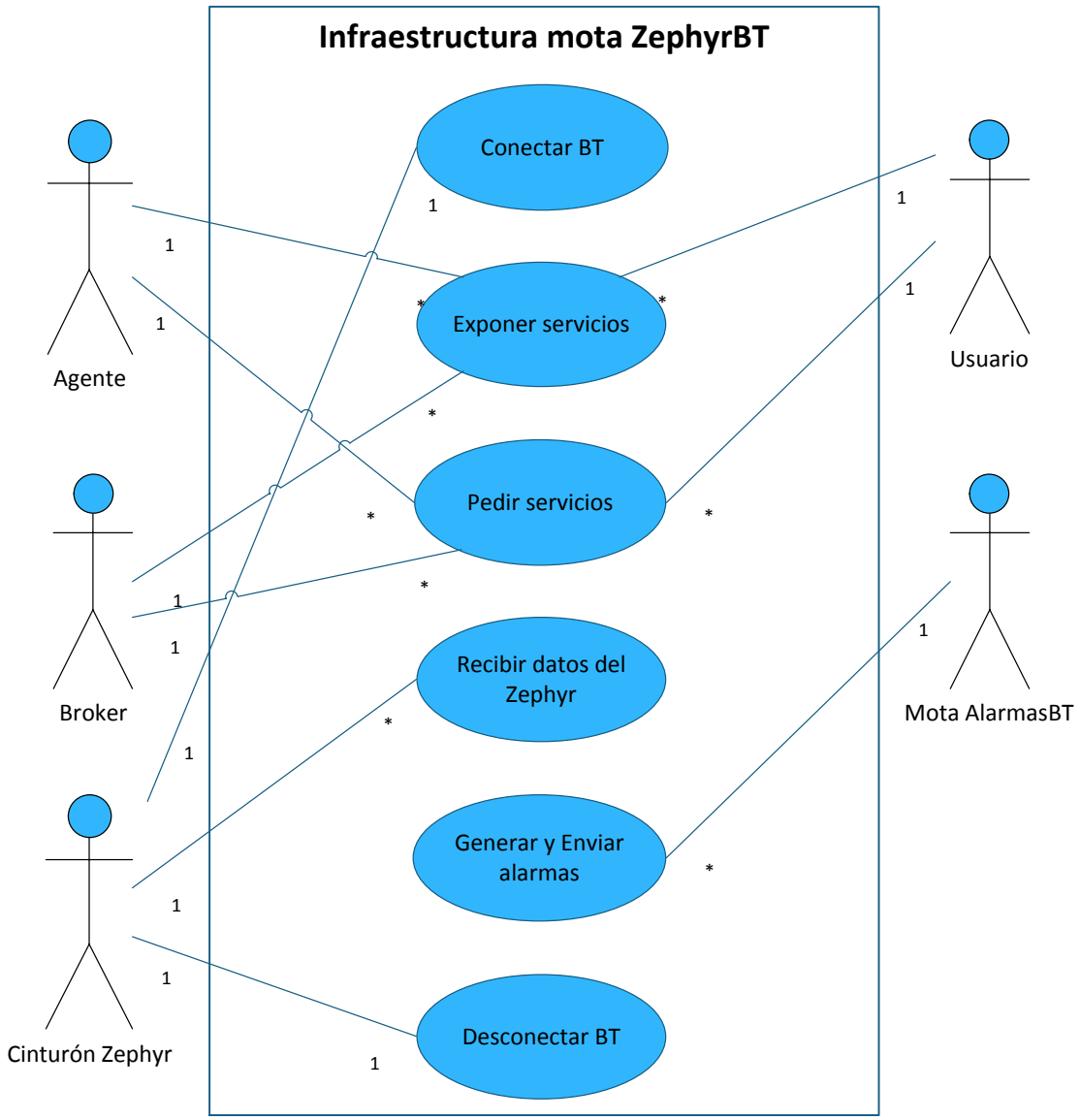


Figura 82: Diagrama UML de casos de uso de la mota ZephyrBT.

5.3.1.2 Explicación de casos de uso

A continuación se muestra una tabla, en la cual se explica cada caso de uso, apareciendo una breve descripción y los actores que están implicados.

CASO DE USO	DESCRIPCIÓN	ACTORES
Conectar BT	La mota ZephyrBT se conecta, vía Bluetooth, al módulo Bioharness Zephyr.	Cinturón Zephyr.
Exponer servicios	Se trata de la publicación de los servicios.	Agente, Broker y Usuario.
Pedir servicios	Cualquiera de los tres actores puede pedir servicios (simples y/o compuestos).	Agente, Broker y Usuario.
Recibir datos Zephyr	El actor Cinturón Zephyr envía parámetros fisiológicos.	Cinturón Zephyr.
Generar y Enviar alarmas	La mota ZephyrBT genera alarmas y se la envía a la otra mota, vía BT.	Mota AlarmasBT
Desconectar BT	Desconexión Bluetooth del enlace establecido entre el nodo Zephyr BT y el cinturón Zephyr.	Cinturón Zephyr.

Tabla 13: Descripción de los casos de uso de la infraestructura mota ZephyrBT.

5.3.2 Infraestructura de la mota AlarmasBT

La infraestructura de la otra mota que dispone de interfaz Bluetooth, denominada AlarmasBT, está compuesta por cuatro casos de uso y dos actores. Los casos de uso son: Conexión BT, Recibir alarmas, Enviar alarmas y Desconexión BT. Por otro lado, los actores son la mota ZephyrBT y el Reloj WiMM. Ver Figura 83.

5.3.2.1 Actores

En esta infraestructura aparecen dos actores:

- *Mota ZephyrBT:* representa a la infraestructura que se ha explicado en el apartado anterior.
- *Reloj WiMM*: se corresponde con el reloj WiMM que dispone de interfaz Bluetooth y se describió en el apartado 2.2.8.4.

5.3.2.2 Explicación de los casos de uso

A continuación, se van a describir el flujo de acciones que tienen lugar en la infraestructura de la mota AlarmasBT.

Lo primero, es establecer una conexión Bluetooth entre el nodo AlarmasBT y el reloj WiMM. Una vez establecida la conexión, se está a la espera de recibir alarmas por parte de la mota ZephryrBt mediante un enlace radio. Cuando se recibe una alarma, se envía a través del

enlace Bluetooth al reloj WiMM. Cuando no queramos transmitir más alarmas al reloj procederemos al cierre del enlace Bluetooth que habíamos establecido.

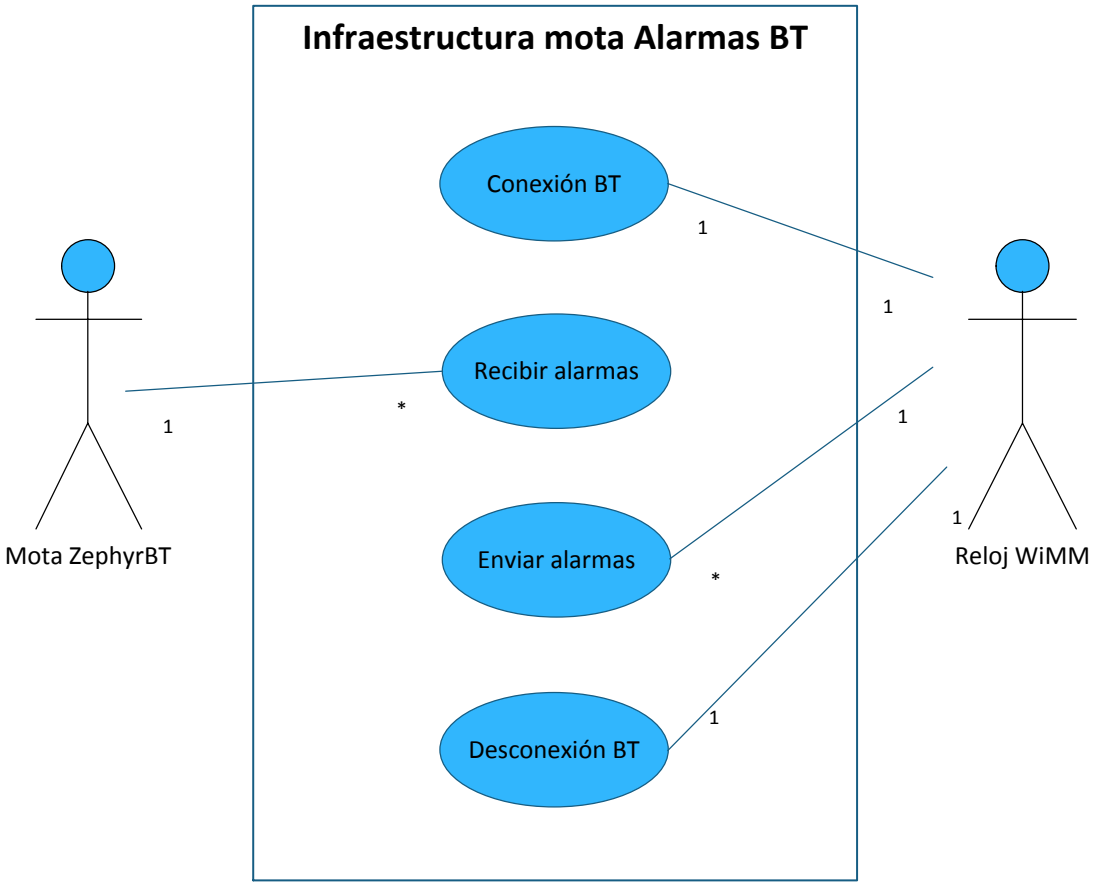


Figura 83: Diagrama UML de casos de uso de la mota Alarmas BT.

5.4 DESCRIPCIÓN DEL ESCENARIO DE LA VALIDACIÓN

Como se indicó al comienzo del capítulo, se trata de un escenario tal como el gimnasio de la Universidad donde interviene un deportista. Para ello se desplegarán una serie de nodos sensores, en particular, tres nodos sensores de temperatura, dos nodos con interfaz Bluetooth (que llevará consigo el deportista), un nodo con la función de Broker, otro implementando el Orquestador y una estación base que hará de Gateway entre el ESB y la red inalámbrica de sensores. También un PC, donde estará alojado el ESB. El deportista, además, llevará el cinturón Bioharness Zephyr y el reloj WiMM.

En la siguiente imagen (Figura 84) se pueden ver las siete motas SunSpot y los dispositivos que lleva el deportista, tanto el módulo Zephyr como el reloj WiMM.

Figura 84: Nodos sensores de la red (Temp1, Tem2, Temp3, Broker y Orquestador), nodos Bluetoooth (ZephyrBT y AlarmasBT), módulo Zephyr y reloj Wimm.

La Figura 85 muestra el escenario desplegado. Como se puede observar, la mota Temperatura1 se encuentra en una de las paredes de la sala principal del gimnasio, junto a uno de los ventanales. La Temperatura2 se encuentra en la sala de pesas y la Temperatura 3 se encuentra en el pasillo, ya en el exterior del gimnasio.

El Orquestador y el Broker se encuentran uno en frente del otro en dos columnas, dentro de la sala principal. El ESB, como ya se mencionó anteriormente, está alojado en un PC, en este caso en un portátil al que está conectado la estación base SunSpot que hace de Gateway. Además hay un punto de acceso para que se pueda acceder al ESB a través de una IP pública desde cualquier lugar y realizar peticiones de los diferentes servicios.

El deportista puede moverse por toda la zona del gimnasio y está equipado con las dos motas con interfaz Bluetooth (ZephyrBT y AlarmasBT), el reloj WiMM y el cinturón Zephyr.

Una vez que se ha desplegado la arquitectura de red, conforme a la Figura 85, procedemos a su activación y puesta en marcha, con el fin de verificar su correcto funcionamiento de todo el sistema. En concreto, analizaremos los parámetros relacionados con la tecnología Bluetooth. Recogiendo información de interés y analizando los resultados obtenidos, los cuales se verán en los siguientes apartados.

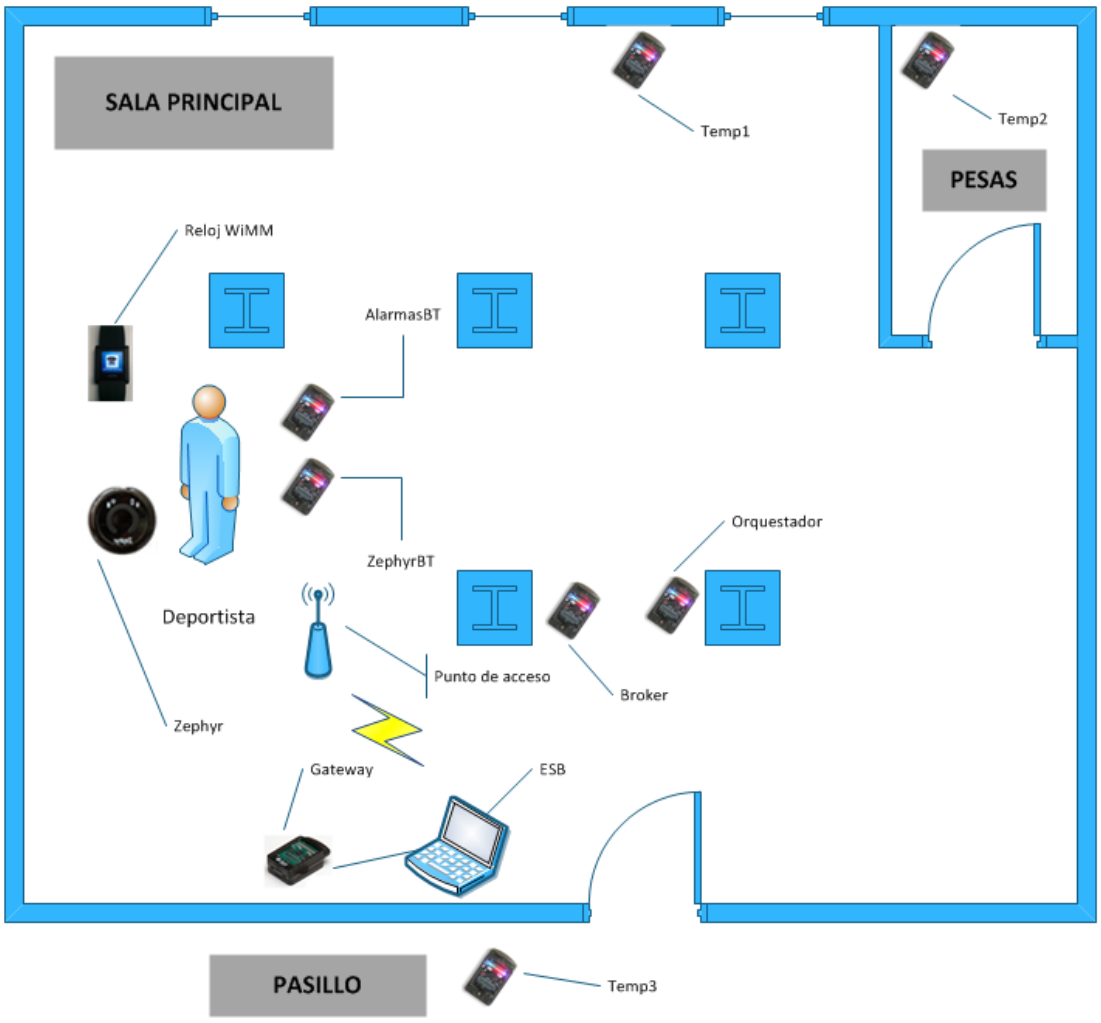


Figura 85: Escenario de validación del sistema.

5.5 ANÁLISIS DE LOS RESULTADOS DE LA VALIDACIÓN

Como ya dijimos en el capítulo 1 de dicha memoria, para realizar un examen exhaustivo del funcionamiento correcto del sistema, vamos a estudiar los resultados obtenidos durante la fase de validación. En concreto: tiempo de conexión entre la mota ZephyrBT y el módulo Bioharness Zephyr, tiempo de desconexión entre la mota ZephyrBT y el módulo Bioharness Zephyr, tiempo de recepción del servicio de temperatura ambiente, tiempo de conexión entre la mota AlarmasBT y el reloj WiMM, tiempo que tarda el Broker en registrar los servicios del Agente de la mota ZephyrBT, y por último el tiempo de recepción de la alarma en el reloj WiMM desde su generación en la mota ZephyrBT.

5.5.1 Tiempo de conexión entre la mota ZephyrBT y el módulo Bioharness Zephyr.

En primer lugar, hemos analizado el tiempo que tarda la mota Zephyr BT en conectarse al módulo Bioharness Zephyr. Entre estos dos dispositivos solo cabe la posibilidad de que la mota inicie la comunicación. Para ello, hemos programado temporizadores, ya que como se vio en el capítulo de iWRAP hay que enviar una serie de comandos AT y esperar las respuestas para establecer la conexión Bluetooth. A continuación, se muestra una gráfica donde se ve ilustrado el tiempo, en segundos, que tarda en conectarse la mota Zephyr BT al módulo Bioharness hasta en 10 ocasiones diferentes.

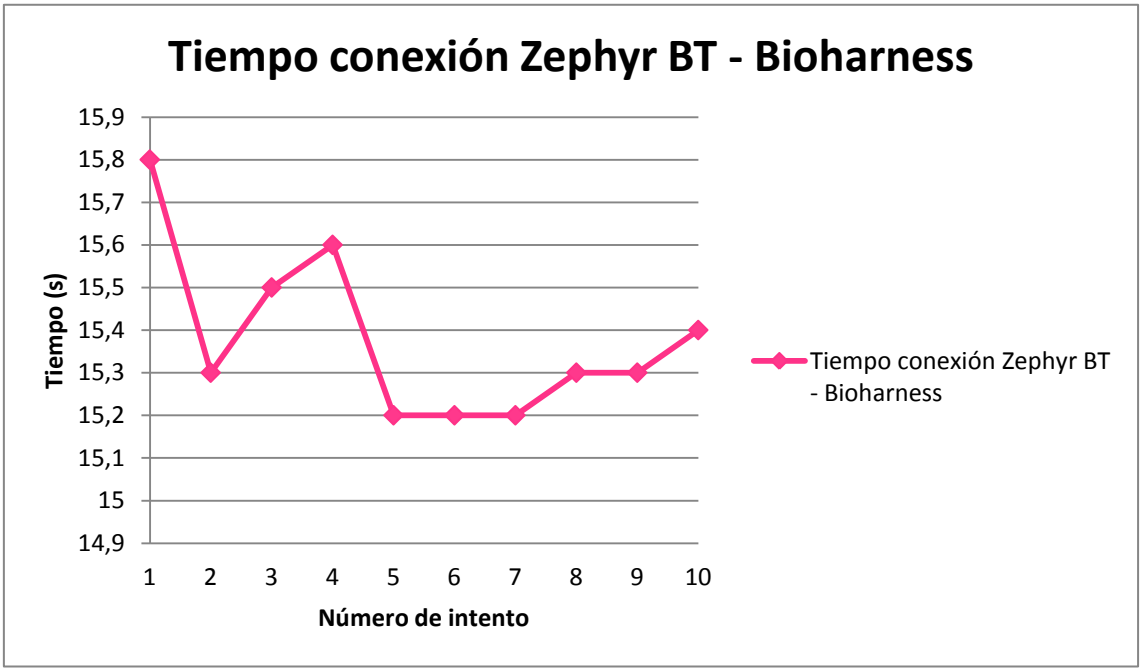


Figura 86: Tiempo de conexión Bluetooth entre la mota Zephyr BT y el módulo Bioharness Zephyr.

En la siguiente tabla, podemos observar el número total de intentos con el tiempo correspondiente que se ha tardado en conectar. También se puede observar la moda y la mediana que coinciden con un valor de 15,3 s; el promedio, es decir, tarda una media de 15,38 s en establecerse el enlace Bluetooth; el tiempo mínimo que ha sido de 15,2 s y el tiempo máximo que son 15,8 s.

	Intentos	Tiempo (s)
	Intento 1	15,8
	Intento 2	15,3
	Intento 3	15,5
	Intento 4	15,6
	Intento 5	15,2
	Intento 6	15,2
	Intento 7	15,2
	Intento 8	15,3
	Intento 9	15,3
	Intento 10	15,4
Moda		15,3
Mediana		15,3
Promedio		15,38
Mínimo		15,2
Máximo		15,8

Tabla 14: Moda, mediana, promedio, mínimo y máximo del tiempo de conexión ZephyrBT – Bioharness.

5.5.2 Tiempo de desconexión entre la mota ZephyrBT y el módulo Bioharness Zephyr.

Después, hemos procedido a analizar los resultados de la desconexión entre la mota ZephyrBT y el módulo Bioharness Zephyr. La desconexión se produce cuando se apaga el módulo Bioharness, pero no es inmediato. Ya que se tarda un tiempo en realizar el liberar en enlace Bluetooth que estaba establecido. Al igual que en el caso anterior, se han realizado un número de 10 intentos y estos han sido los resultados:

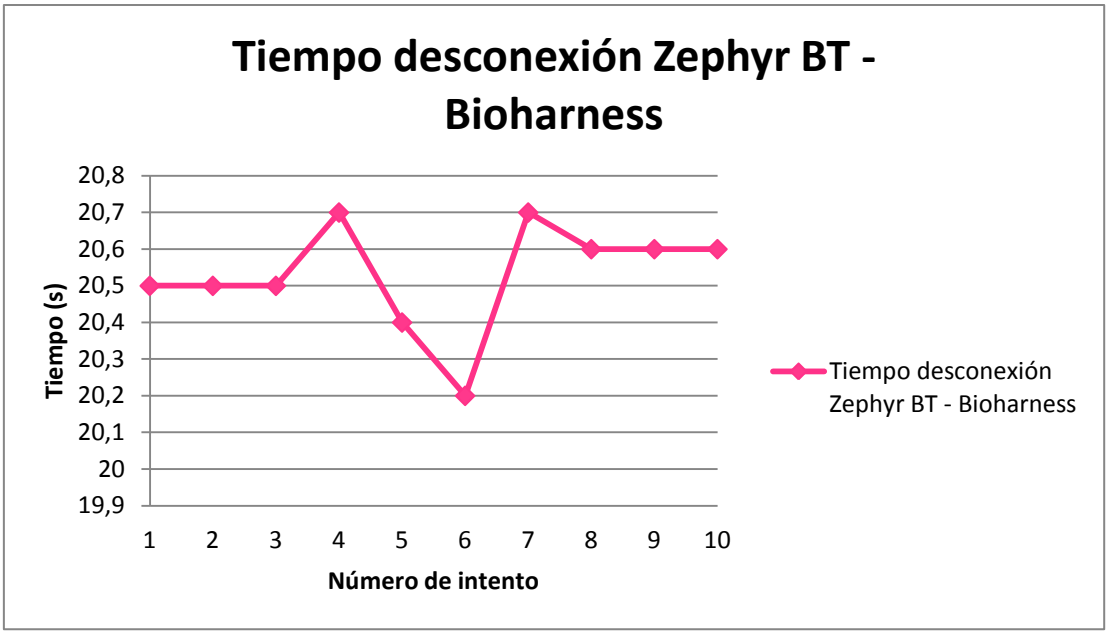


Figura 87: Tiempo de desconexión Bluetooth entre la mota Zephyr BT y el módulo Bioharness Zephyr.

En la siguiente tabla se ve reflejado el número de intentos con el tiempo correspondiente, en segundos, que ha tardado la mota ZephyrBT en conectarse al Módulo Bioharness, así como la moda con un valor de 20,5 s y la mediana con un valor de 20,55 s; el promedio con un valor de 20,53 s; el tiempo mínimo que han sido 20,2 s y el tiempo máximo que son 20,7 s.

Intentos	Tiempo (s)
Intento 1	20,5
Intento 2	20,5
Intento 3	20,5
Intento 4	20,7
Intento 5	20,4
Intento 6	20,2
Intento 7	20,7
Intento 8	20,6
Intento 9	20,6
Intento 10	20,6
Moda	20,5
Mediana	20,55
Promedio	20,53
Mínimo	20,2
Máximo	20,7

Tabla 15: Moda, mediana, promedio, mínimo y máximo del tiempo de desconexión ZephyrBT – Bioharness.

5.5.3 Tiempo de recepción del servicio de temperatura ambiente.

En este apartado vamos a analizar el tiempo que tarda nuestro agente en recoger el dato de la temperatura ambiente a través del servicio *catchTemperature*, que en nuestro caso, lo proporciona la mota Temperatura 2. Como se observa en la siguiente gráfica el valor varía más que en los casos anteriores, ya que este servicio no depende de la conexión Bluetooth. Nosotros recibiremos un dato, aproximadamente cada 25 segundos, pero en ocasiones si el Broker está atendiendo otras peticiones nos tardará más o bien si el Broker está libre tardará menos. Veamos los resultados:

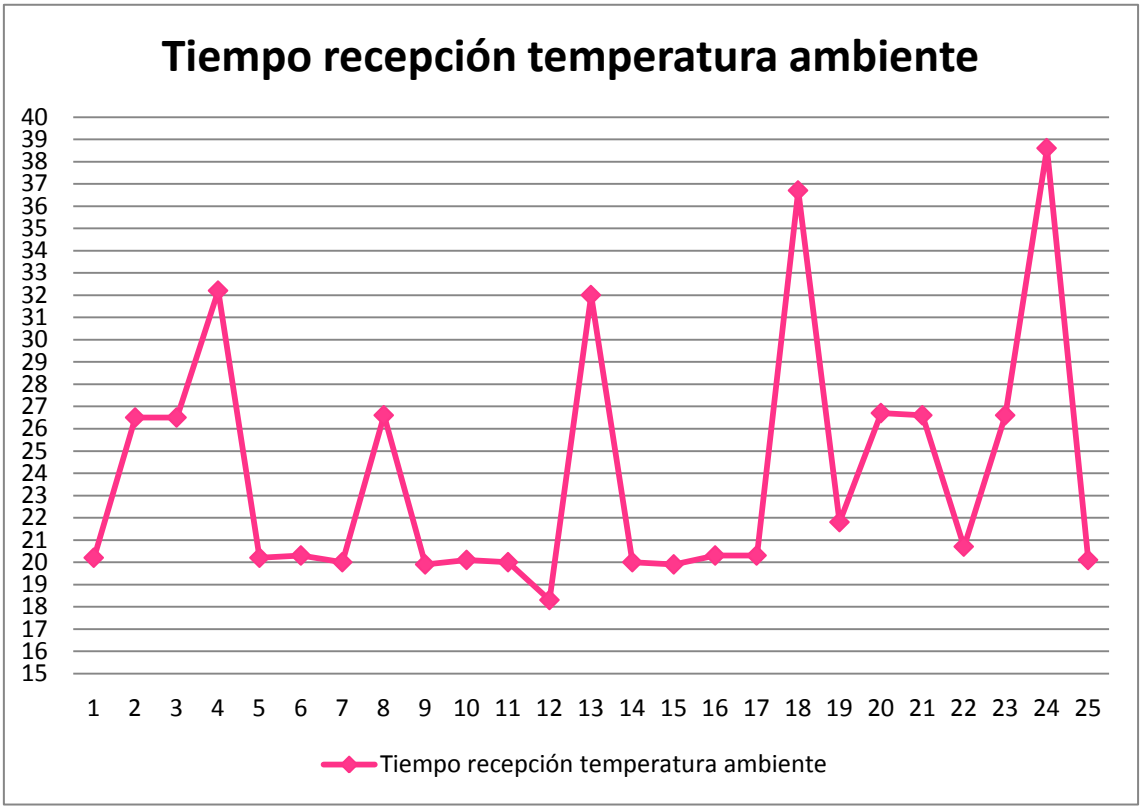


Figura 88: Tiempo que tarda en recibir el agente ZephyrBT la temperatura ambiente.

En la tabla de la izquierda, se puede observar el número total de intentos con el tiempo exacto que se ha tardado en recibir la temperatura. Por otro lado, en la tabla e la derecha se puede observar la moda y la mediana que coinciden con un valor de 20,3 s; el promedio que es de 24,044s; el tiempo mínimo en recibir la temperatura que ha sido de 18,3 s y el tiempo máximo que son 38,6 s.

Intento	Tiempo (s)
1	20,2
2	26,5
3	26,5
4	32,2
5	20,2
6	20,3
7	20,0
8	26,6
9	19,9
10	20,1
11	20,0
12	18,3
13	32,0
14	20,0
15	19,9
16	20,3
17	20,3
18	36,7
19	21,8
20	26,7
21	26,6
22	20,7
23	26,6
24	38,6
25	20,1

Moda		20,3
Mediana		20,3
Promedio		24,044
Mínimo		18,3
Máximo		38,6

Tabla 16: A la izquierda, tabla de intentos con el tiempo correspondiente de lo que se tarda en recibir la temperatura ambiente. A la derecha, moda, mediana, promedio, mínimo y máximo de los valores anteriores.

5.5.4 Tiempo de conexión entre la mota AlarmasBT y el reloj WiMM

A continuación, vamos a ver el tiempo que tarda la mota encargada de recibir las alarmas, llama Alarmas BT, en conectarse al reloj WiMM. Al igual que en la otra conexión Bluetooth se han establecido una serie de temporizadores para establecer el enlace. Nosotros hemos elegido que sea la mota quien inicia la comunicación y el reloj se mantenga a la espera. También se podría haber implementado al contrario, es decir que fuese el reloj quién iniciase la comunicación. Veamos el tiempo que tarda en establecerse el enlace Bluetooth en 10 pruebas distintas:

Intento	1	2	3	4	5	6	7	8	9	10
Tiempo (s)	35,8	35,5	35,8	35,4	35,3	35,1	35,3	35,4	35,1	35,3

Moda	Mediana	Prom.	Min.	Max.
35,3	35,35	35,4	35,1	35,8

Tabla 17: Tiempo que tarda en conectarse la mota AlarmasBT al reloj WiMM, moda, media, promedio, tiempo mínimo y máximo en segundos.

Además, se puede observar como el tiempo mínimo en establecerse el enlace bidireccional Bluetooth ha sido de 35,1 segundos y el máximo 35,8 segundos; obteniendo un promedio de 35, 4 segundos.

5.5.5 Registro de los servicios ofrecidos por el agente de la mota ZephyrBT

Como se explicó en apartados anteriores, la mota ZephyrBT dispone de un Agente llamado nSOMZephyrAgent, que ofrece cuatro servicios de temperatura corporal, temperatura ambiente, ritmo cardiaco y tasa respiratoria. En este caso hemos realizado un total de 10 mediciones a cerca de cuánto tiempo tarda el Broker en registrar los servicios de dicho Agente. Veamos la siguiente gráfica:

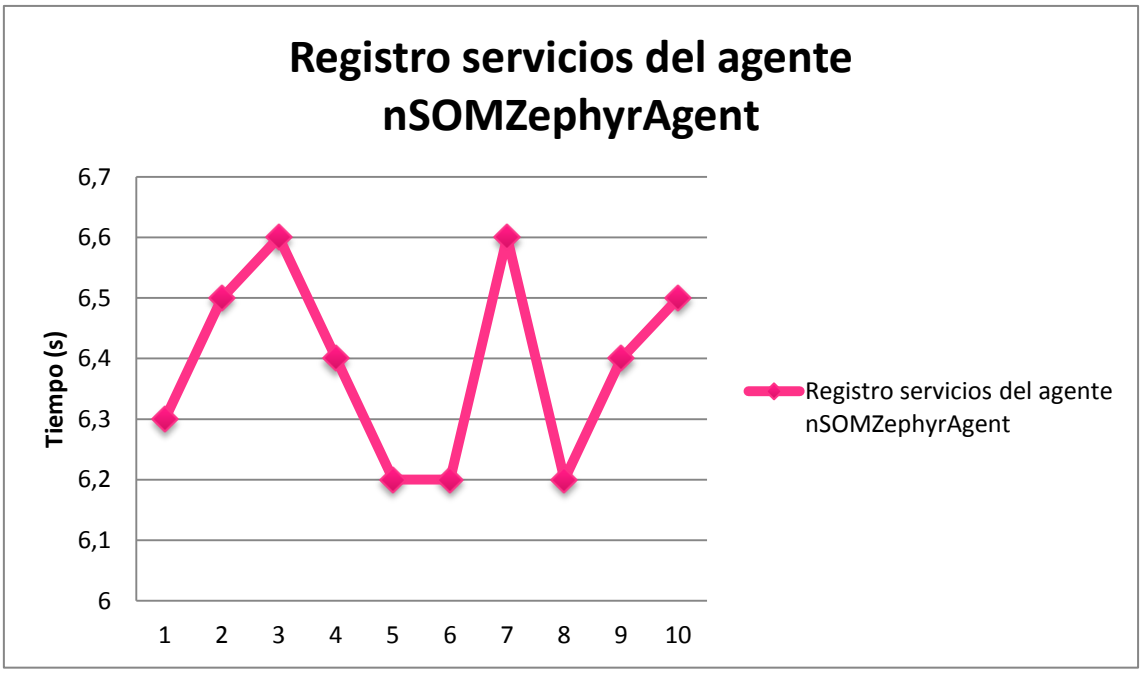
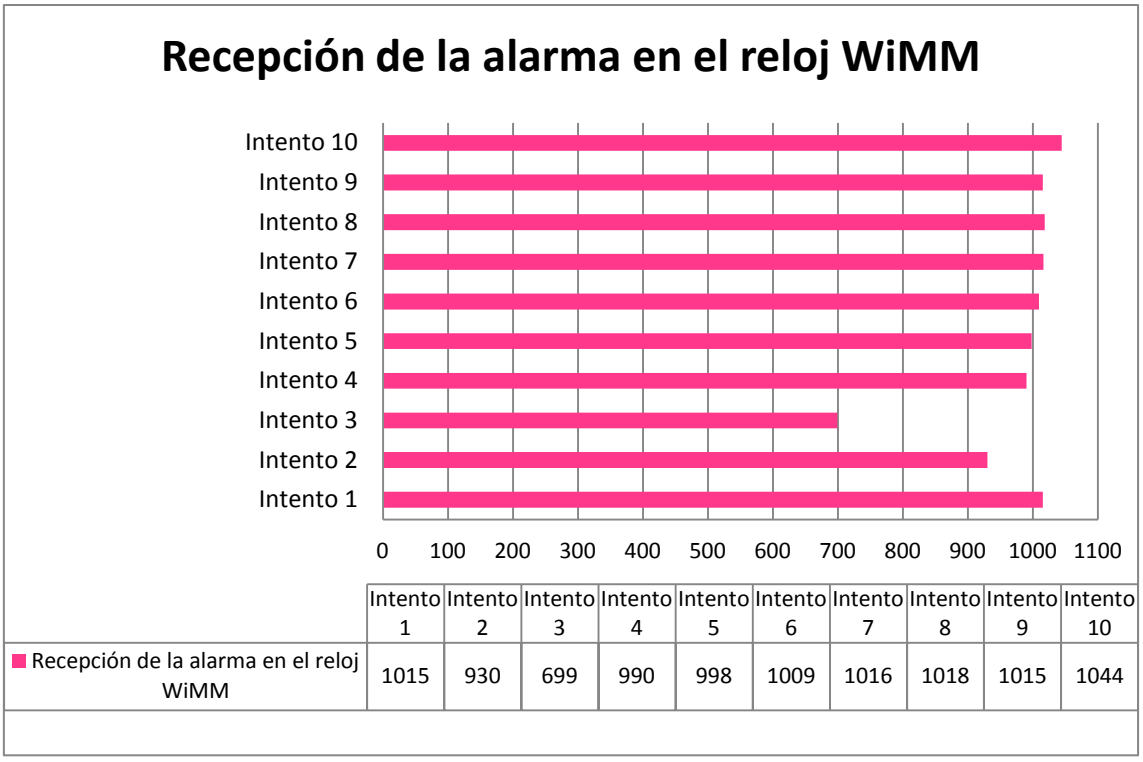


Figura 89: Tiempo de registro de los servicios ofrecidos por el agente NSOMZephyrAgent.

5.5.6 Recepción de la alarma en el reloj WiMM

Por último, en este apartado vamos a estudiar el tiempo que se tarda en mostrar una alarma en el reloj WiMM que lleva consigo el deportista desde que se genera en la mota Zephyr BT. Como se explicó en capítulos anteriores, una vez generada la alarma, se transmite vía radio a la mota Alarmas BT y esta actúa de pasarela para enviársela, a través de la interfaz Bluetooth al módulo WiMM. En la siguiente gráfica se puede observar lo citado anteriormente, en un total de 10 intentos y el tiempo que se ha tardado expresado en milisegundos.



	Intento 1	Intento 2	Intento 3	Intento 4	Intento 5	Intento 6	Intento 7	Intento 8	Intento 9	Intento 10
Recepción de la alarma en el reloj WiMM	1015	930	699	990	998	1009	1016	1018	1015	1044

Figura 90: Tiempo de recepción de la alarma en el reloj WiMM.

Analizando los datos, podemos concluir que el tiempo medio desde que se genera una alarma hasta que el deportista lo visualiza en su reloj es de 973,4 milisegundos. Este valor es aceptable, ya que al ser inferior a 1 segundo la generación de alarmas son fiables e instantáneas.

Capítulo 6

Conclusiones y trabajos futuros

6.1 CONCLUSIONES

El contenido de esta memoria se ha centrado en cuatro bloques fundamentales: un estudio del Estado del Arte en Redes Inalámbricas de Sensores, que no sirve para afrontar el resto de contenido de la memoria. En este mismo bloque se incluye el estándar Bluetooth, cuya importancia es muy grande a lo largo del proyecto. El siguiente bloque es el firmware iWRAP, donde se explican sus modos de operación, comandos y mensajes de error. El tercer bloque abarca el capítulo 4, donde se explica la arquitectura y especificación NSOM. Por último, otro bloque dedicado a una fase de validación del sistema en un escenario de aplicación real, donde se pueden ver representados los datos analizados.

Para concluir esta memoria, se realizará un breve resumen de cada uno de los capítulos abordados en este Proyecto Fin de Carrera. Así, conseguiremos extraer algunas conclusiones que nos permitan valorar el trabajo realizado. También se aportarán algunas ideas para trabajos futuros de investigación haciendo uso de dicha tecnología.

En el **capítulo 1**, se hizo una pequeña introducción a las Redes Inalámbricas de Sensores. También se habló de los objetivos del Proyecto Fin de Carrera y de su organización en dicha memoria. A la vez, se menciona en qué contexto está enmarcado dicho proyecto.

En el **capítulo 2**, titulado Estado del Arte en Redes Inalámbricas de Sensores, se habla precisamente del Estado del Arte, describiendo su arquitectura y los componentes que lo forman. Además se realiza una descripción en detalle del estándar Bluetooth. En el cual, se habla de sus orígenes e historia, de su alcance y de las diferentes topologías de red que se pueden configurar.

Además, se mencionan todas las especificaciones Bluetooth que han ido apareciendo desde el momento que apareció la tecnología Bluetooth. También se describen cada uno de los perfiles Bluetooth, describiendo la arquitectura de protocolos del perfil SPP que ha sido el que se ha utilizado a lo largo del Proyecto.

Por último, en este capítulo se habla de los dispositivos Bluetooth que se han utilizado a lo largo de dicho Proyecto.

En el **capítulo 3**, se trata el firmware iWRAP v4. Primero, hemos realizado una breve introducción para enlazar con el resto de apartados. A continuación, hemos mencionado los modos iWRAP, describiendo cada uno de ellos. Después se han descrito los comandos AT más

importantes y los que hemos usado para la configuración de nuestros módulos Bluetooth.

En otro apartado, hemos hablado de los posibles mensajes de error que pueden surgir en dicho firmware, y por último, se han puesto dos ejemplos de conexión Bluetooth empleando el perfil SPP.

En el **capítulo 4**, se ha abordado la arquitectura y especificación nSOM. Hemos empezado definiendo el concepto de middleware para pasar a la explicación del paradigma de programación orientado a servicios (SOC). Una vez explicados estos conceptos, hemos hablado de la arquitectura orientada a servicios (SOA). A continuación, se ha descrito la arquitectura específica de nuestro Proyecto, conocida como nSOM. Dentro de dicha arquitectura se han explicado tanto las diferentes plataformas que lo forman como sus componentes. También se mencionan los servicios que puede ofrecer dicho middleware, a través de una clasificación por niveles.

En otro apartado, se describe el concepto de Agente y se describen específicamente los que hemos utilizado en nuestro proyecto. En nuestro middleware nSOM hemos utilizado el formato JSON, por lo que también se hace una breve descripción sobre él para introducirnos en SMD, una representación JSON que describe servicios web. A continuación se explica el framework RDF que se usa en nuestro servicio de descripción, descubrimiento e invocaciones que utiliza un lenguaje ligero llamado nSOL.

También hablamos del subsistema orientado a servicios, compuesto por el Broker y el Orquestador. Describiendo cada uno de estos componentes y explicando su función. Otro elemento importante dentro de nuestra arquitectura nSOM es el ESB, que es la columna vertebral de nuestra implementación SOA.

En este mismo capítulo, describimos los tipos de nodos que hay dentro de la red. Se pueden clasificar en nodos con agentes ofreciendo servicios de temperatura y nodos con Bluetooth.

Después, en otro apartado distinto, hablamos del modelo de comunicación. Donde se distingues los servicios simples de los servicios compuestos, donde se ponen varios ejemplos para ver y entender su funcionamiento.

A continuación se realiza una descripción del diseño detallado de la arquitectura, donde se hablará en concreto del nodo Bluetooth ZephyrBT y del otro nodo AlarmasBT.

Por último, en el **capítulo 5**, titulado Validación del sistema y análisis de resultados, nos centramos primeramente en la justificación del escenario planteado. Es un escenario del deportista, y se realiza en el gimnasio de nuestra escuela. Se medirán diferentes resultados con respecto a las interfaces Bluetooth que es la parte que nos concierne del proyecto.

A continuación, se describirán los diferentes casos de uso que se plantean en dicho escenario. Se abordará el caso de uso en la infraestructura de la mota ZephyrBT, donde se explicarán los diferentes casos de uso y los actores que intervienen. El otro caso de uso abordado, es la infraestructura formada por la mota AlarmasBT, que al igual que en el caso anterior, se dará una explicación de los diferentes casos de uso que se plantean y de los actores que intervienen.

En otro apartado se describe el escenario de validación, donde se enumeran y explican los elementos que forman nuestra arquitectura así como su distribución y posicionamiento dentro del gimnasio.

6.2 TRABAJOS FUTUROS

Una vez dadas las conclusiones y el trabajo específico que se ha realizado durante este Proyecto Fin de Carrera, se han definido una serie de mejoras o trabajos futuros que pueden ayudan, tanto a optimizar el sistema, como incluir mejoras en el mismo:

- Emplear una única mota SunSpot con interfaz Bluetooth que sustituya a las 2 que tenemos. Para ello tendríamos que emplear la UART y la USART que nos proporcionan dichos sensores inalámbricos, así como las dos placas Sparkfun Bluegiga en la misma mota. Ya que, como se mencionó en capítulos anteriores, en nuestro caso solo se puede establecer una conexión Bluetooth con perfil SPP punto a punto.

- Obtener el posicionamiento del deportista, es decir, saber en cada momento en que parte o habitación del gimnasio se encuentra. Para ello utilizaríamos nodos Bluetooth dispersos por el gimnasio, para así saber dónde está el deportista. Por ejemplo, si el usuario está en la sala de pesas, el Blueooth del nodo que lleva consigo se comunicaría con el nodo Bluetooth que está posicionado en dicha habitación. Esta información nos permitiría saber en qué lugar exacto estaría el deportista.

- Monitorización de más parámetros provenientes del módulo Zephyr Bioharness. Como

es el caso del electrocardiograma, que permitiría ver una gráfica en tiempo real los latidos del corazón del deportista.

Anexo I

API de la arquitectura del nodo ZephyrBT

org.diyse.nSOMAgent

Interface nSOMAgent

All Known Implementing Classes:

nSOMAccelerometerAgent, nSOMLEDAgent, nSOMLightAgent, nSOMTemperatureAgent, nSOMZephyrAgent

---

```
public interface nSOMAgent
```

Project: LifeWear

Method Summary

java.lang.String	getnSOMAgentDescription() Obtain the nSOMAgentDescription.
nSOMServiceAgentObject	getServiceAgentObject() Obtain the service description objetc for this nSOMLightAgent.
void	load(nSOMContext nSOMContext) Performs the load process of the nSOMAgent in the Service Execution Platform.
void	receptacle(ServiceContext operationInvocation) Implements the receptacle of the nSOMAgents.
void	run() Performs the run process of the nSOMAgent and the service registering in the Brokers.
void	stop() Performs the stop process of the nSOMAgent and the service unregistering in the Brokers.
void	unload() Performs the unload process of the nSOMAgent in the Service Execution Platform.

Method Detail

load

```
void load(nSOMContext nSOMContext)
```

Performs the load process of the nSOMAgent in the Service Execution Platform.

Parameters:

`Context` - provided by the nSOMContainer

---

run

```
void run()
```

Performs the run process of the nSOMAgent and the service registering in the Brokers.

---

stop

```
void stop()
```

Performs the stop process of the nSOMAgent and the service unregistering in the Brokers.

---

unload

```
void unload()
```

Performs the unload process of the nSOMAgent in the Service Execution Platform.

---

receptacle

```
void receptacle(ServiceContext operationInvocation)
```

Implements the receptacle of the nSOMAgents. This method is invoked to pass the service invocations to the agents.

Parameters:

`ServiceContext` - of the service invocation

---

getServiceAgentObject

```
nSOMServiceAgentObject getServiceAgentObject()
```

Obtain the service description objetc for this nSOMLightAgent.

Returns:

nSOMServiceAgentObject of the nSOMLightAgent

---

getnSOMAgentDescription

```
java.lang.String getnSOMAgentDescription()
```

Obtain the nSOMAgentDescription. This method provides the name of the agent which have been invoked.

Returns:

String with the agent description

---

org.diyse.nSOMAgent

Class nSOMZephyrAgent

```
java.lang.Object
  └ org.diyse.nSOMAgent.nSOMZephyrAgent
```

All Implemented Interfaces:

nSOMAgent

```
public class nSOMZephyrAgent
extends java.lang.Object
implements nSOMAgent
```

Project: LifeWear

Constructor Summary

nSOMZephyrAgent() Constructor of the nSOMZephyrAgent Class.

Method Summary

void	catchTemperature(int temp) Measure the environmental temperature
double	getBodyTemperature() Measure the body temperature
double	getBreathingRate() Measure the breathing rate
int	getHeartRate() Measure the heart rate
java.lang.String	getnSOMAgentDescription() Obtain the nSOMAgentDescription.
nSOMServiceAgentObject	getServiceAgentObject() Obtain the service description object for this nSOMZephyrAgent.
void	load(nSOMContext nSOMContext) Performs the load process of the nSOMZephyrAgent in the Service Execution Platform.
void	receptacle(ServiceContext operationInvocation) Implements the receptacle of the nSOMZephyrAgent.
void	run() Performs the run process of the nSOMZephyrAgent and the services registering in the Brokers.
void	stop() Performs the stop process of the nSOMZephyrAgent and the services unregistering in the Brokers.
void	unload() Performs the unload process of the nSOMZephyrAgent in the Service Execution Platform.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

nSOMZephyrAgent

```
public nSOMZephyrAgent()
```

Constructor of the nSOMZephyrAgent Class.

Method Detail

load

```
public void load(nSOMContext nSOMContext)
```

Performs the load process of the nSOMZephyrAgent in the Service Execution Platform.

Specified by:

load in interface nSOMAgent

Parameters:

Context - provided by the nSOMContainer

---

run

```
public void run()
```

Performs the run process of the nSOMZephyrAgent and the services registering in the Brokers.

Specified by:

run in interface nSOMAgent

---

stop

```
public void stop()
```

Performs the stop process of the nSOMZephyrAgent and the services unregistering in the Brokers.

Specified by:

stop in interface nSOMAgent

---

unload

```
public void unload()
```

Performs the unload process of the nSOMZephyrAgent in the Service Execution Platform.

Specified by:
`unload` in interface `nSOMAgent`

Parameters:
`Context` - provided by the nSOMContainer

---

getHeartRate

```
public int getHeartRate()
```

Measure the heart rate

Returns:
the heart rate

---

getBreathingRate

```
public double getBreathingRate()
```

Measure the breathing rate

Returns:
the breathing rate

---

getBodyTemperature

```
public double getBodyTemperature()
```

Measure the body temperature

Returns:
the body temperature

---

catchTemperature

```
public void catchTemperature(int temp)
```

Measure the environmental temperature

Parameters:
`temp` - the environmental temperature

---

receptacle

```
public void receptacle(ServiceContext operationInvocation)
```

Implements the receptacle of the nSOMZephyrAgent. This method is invoked to pass the service invocations to the agents.

Specified by:

receptacle in interface nSOMAgent

Parameters:

ServiceContext - of the service invocation

---

getServiceAgentObject

```
public nSOMServiceAgentObject getServiceAgentObject()
```

Obtain the service description object for this nSOMZephyrAgent.

Specified by:

getServiceAgentObject in interface nSOMAgent

Returns:

nSOMServiceAgentObject of the nSOMNoiseAgent

---

getnSOMAgentDescription

```
public java.lang.String getnSOMAgentDescription()
```

Obtain the nSOMAgentDescription. This method provides the name of the Zephyr agent which have been invoked.

Specified by:

getnSOMAgentDescription in interface nSOMAgent

Returns:

String with the agent description

---

org.diyse.hardware.sensor

Class AlarmsDetection

```
java.lang.Object
  └ org.diyse.hardware.sensor.AlarmsDetection
```

```
public class AlarmsDetection
extends java.lang.Object
```

Project: LifeWear

Constructor Summary

AlarmsDetection()

Method Summary

void	checkBodyTempAlarm(double bodyTemp) Check if there is body temperature alarm.
void	checkEnvTempAlarm(int envTemp) Check if there is environmental temperature alarm.
void	checkHeartRateAlarm(int heartRate) Check if there is heart rate alarm.
boolean	getOpenPort() Resolve if a port is opening
void	openPort() Open the port 70 to send information about alarms

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

AlarmsDetection

```
public AlarmsDetection()
```

Method Detail

openPort

```
public void openPort()
```

Open the port 70 to send information about alarms

---

checkEnvTempAlarm

```
public void checkEnvTempAlarm(int envTemp)
```

Check if there is environmental temperature alarm. And if there is alarm, it has been sent.

Parameters:

`envTemp` - the environmental temperature.

---

checkBodyTempAlarm

```
public void checkBodyTempAlarm(double bodyTemp)
```

Check if there is body temperature alarm. And if there is alarm, it has been sent.

Parameters:

`bodyTemp` - the body temperature

---

checkHeartRateAlarm

```
public void checkHeartRateAlarm(int heartRate)
```

Check if there is heart rate alarm. And if there is alarm, it has been sent.

Parameters:

`heartRate` - the heart rate

---

getOpenPort

```
public boolean getOpenPort()
```

Resolve if a port is opening

Returns:

True if there is a port opening; False in other case

---

org.diyse.hardware.sensor

Class BioharnessDevice

```
java.lang.Object
  └org.diyse.hardware.sensor.BioharnessDevice
```

```
public class BioharnessDevice
extends java.lang.Object
```

Project: LifeWear

Field Summary

static int	BREATHING_RATE_LS
static int	BREATHING_RATE_MS
static int	CORE_TEMPERATURE_LS
static int	CORE_TEMPERATURE_MS
static int	ENABLE
static byte	ETX_b
static int	HEART_RATE_LS
static int	ID_SUMMARY_DATA_PACKET
static int	ROG
static int	STX
static byte	STX_b
static int	SUMMARY_PACKET_SIZE

Constructor Summary

BioharnessDevice(java.io.DataInputStream input, java.io.DataOutputStream output)
Creates a BioharnessDevice instance

Method Summary	
double	getBodyTemperature() Get the body temperature
double	getBreathingRate() Get the breathing rate
int	getHeartRate() Get heart rate
int	mergeUnsigned(int low, int high) Merge two int
boolean	readBuffer(long ms) Read the data from de buffer
java.lang.String	readString(long ms) Read a String from the buffer
void	setupSummaryBioharness() Set up Summary Bioharness Packet

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

ETX_b

```
public static final byte ETX_b
```

See Also:

Constant Field Values

STX_b

```
public static final byte STX_b
```

See Also:

Constant Field Values

STX

```
public static final int STX
```

See Also:

Constant Field Values

ENABLE

```
public static final int ENABLE
```

See Also:
Constant Field Values

ID_SUMMARY_DATA_PACKET

```
public static final int ID_SUMMARY_DATA_PACKET
```

See Also:
Constant Field Values

ROG

```
public static final int ROG
```

See Also:
Constant Field Values

HEART_RATE_LS

```
public static final int HEART_RATE_LS
```

See Also:
Constant Field Values

BREATHING_RATE_LS

```
public static final int BREATHING_RATE_LS
```

See Also:
Constant Field Values

BREATHING_RATE_MS

```
public static final int BREATHING_RATE_MS
```

See Also:
Constant Field Values

CORE_TEMPERATURE_LS

```
public static final int CORE_TEMPERATURE_LS
```

See Also:
Constant Field Values

CORE_TEMPERATURE_MS

```
public static final int CORE_TEMPERATURE_MS
```

See Also:

Constant Field Values

SUMMARY_PACKET_SIZE

```
public static final int SUMMARY_PACKET_SIZE
```

See Also:

Constant Field Values

Constructor Detail

BioharnessDevice

```
public BioharnessDevice(java.io.DataInputStream input,
                        java.io.DataOutputStream output)
```

Creates a BioharnessDevice instance

Parameters:

`input` - DataInputStream to read information from Zephyr device
`output` - DataOutputStream to send information from Zephyr device

Method Detail

readBuffer

```
public boolean readBuffer(long ms)
```

Read the data from de buffer

Parameters:

`ms` - time to sleep

Returns:

True if the bluetooth is connected; False if it is disconnected

---

readString

```
public java.lang.String readString(long ms)
```

Read a String from the buffer

Parameters:

`ms` - time to sleep

Returns:

a String with the response

---

setupSummaryBioharness

```
public void setupSummaryBioharness()
                            throws java.io.IOException
```

Set up Summary Bioharness Packet

Throws:

`java.io.IOException`

---

mergeUnsigned

```
public int mergeUnsigned(int low,
                         int high)
```

Merge two int

Parameters:

`low` - value

`high` - value

Returns:

the merge result

---

getHeartRate

```
public int getHeartRate()
```

Get heart rate

Returns:

the heart rate

---

getBreathingRate

```
public double getBreathingRate()
```

Get the breathing rate

Returns:

the breathing rate

---

getBodyTemperature

```
public double getBodyTemperature()
```

Get the body temperature

Returns:

the body temperature

---

org.diyse.hardware.sensor

Class BluetoothDriver

```
java.lang.Object
  └org.diyse.hardware.sensor.BluetoothDriver
```

```
public class BluetoothDriver
extends java.lang.Object
```

Project: LifeWear

Constructor Summary

`BluetoothDriver()` Creates a BluetoothDriver instance

Method Summary

void	`connectBTDriver()` Connect the Bluetooth Driver
BioharnessDevice	`getBioharnessDevice()`
boolean	`getConnected()`

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

BluetoothDriver

```
public BluetoothDriver()
```

Creates a BluetoothDriver instance

Method Detail

connectBTDriver

```
public void connectBTDriver()
```

Connect the Bluetooth Driver

---

getConnected

```
public boolean getConnected()
```

Returns:

True if the driver BT is connected. False if is not connected

---

getBioharnessDevice

```
public BioharnessDevice getBioharnessDevice()
```

Returns:

The Zephyr bioharness device

---

org.diyse.hardware.sensor

Class Leds

```
java.lang.Object
  └org.diyse.hardware.sensor.Leds
```

```
public class Leds
extends java.lang.Object
```

Project: LifeWear

Constructor Summary

Leds() Creates a Leds instance

Method Summary

void	TurnOnLed(LEDColor color, int times, int position) Turn on one led of the SunSpot
void	TurnOnLeds(LEDColor color, long ms) Turn on the leds of the SunSpot

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Leds

```
public Leds()
```

Creates a Leds instance

Method Detail

TurnOnLeds

```
public void TurnOnLeds(LEDColor color,
                       long ms)
```

Turn on the leds of the SunSpot

Parameters:

`color` - the color of the leds

`ms` - the time that leds are turning on

TurnOnLed

```
public void TurnOnLed(LEDColor color,
                      int times,
                      int position)
```

Turn on one led of the SunSpot

Parameters:

`color` - the color of the leds
`times` - the time that the led is turning on
`position` - the position on the led that is turning on

Anexo II

API de la arquitectura del nodo AlarmasBT

org.btwimm

Class Leds

```
java.lang.Object
  └org.btwimm.Leds
```

```
public class Leds
extends java.lang.Object
```

Project: LifeWear

Constructor Summary

Leds() Creates a Leds instance

Method Summary

void	TurnOnFirstLed(LEDColor color, long ms) Turn on the first led of the SunSpot
void	TurnOnLed(LEDColor color, int times, int position) Turn on one led of the SunSpot
void	TurnOnLeds(LEDColor color, long ms) Turn on the leds of the SunSpot

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Leds

```
public Leds()
```

Creates a Leds instance

Method Detail

TurnOnLeds

```
public void TurnOnLeds(LEDColor color,
                       long ms)
```

Turn on the leds of the SunSpot

Parameters:

`color` - the color of the leds

`ms` - the time that leds are turning on

TurnOnLed

```
public void TurnOnLed(LEDColor color,
                      int times,
                      int position)
```

Turn on one led of the SunSpot

Parameters:

`color` - the color of the leds
`times` - the time that the led is turning on
`position` - the position on the led that is turning on

TurnOnFirstLed

```
public void TurnOnFirstLed(LEDColor color,
                           long ms)
```

Turn on the first led of the SunSpot

Parameters:

`color` - the color of the led
`ms` - the time that the led is turning on

---

org.btwimm

Class SunSpotApplication

```
java.lang.Object
  └MIDlet
      └org.btwimm.SunSpotApplication
```

```
public class SunSpotApplication
extends MIDlet
```

Project: LifeWear

Constructor Summary

SunSpotApplication()

Method Summary

protected void	destroyApp(boolean unconditional)
protected void	pauseApp()
java.lang.String	readString(long ms) Read a string
protected void	startApp() This class is a bridge between the Zephyr BT and the Wimm watch.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

SunSpotApplication

```
public SunSpotApplication()
```

Method Detail

startApp

```
protected void startApp()
                  throws MIDletStateChangeException
```

This class is a bridge between the Zephyr BT and the Wimm watch. It receives an alarm through the radio interface from the SunSpot called Zephyr BT and send it by Bluetooth to the Wimm watch.

Throws:

MIDletStateChangeException

---

pauseApp

```
protected void pauseApp()
```

---

destroyApp

```
protected void destroyApp(boolean unconditional)
                   throws MIDletStateChangeException
```

Parameters:

`unconditional` -

Throws:

`MIDletStateChangeException`

---

readString

```
public java.lang.String readString(long ms)
```

Read a string

Parameters:

`ms` - the time to sleep

Returns:

the read string

Referencias bibliográficas

[Augusto&] Juan Carlos Augusto, Hideyuki Nakashima y Hamid Aghajan. "*Ambient Intelligence and Smart Environments: A State of the Art*". [Artículo].

[Bioharness API 11] "*Bioharness Bluetooth API Guide*". Zephyr Technology. [En línea]. Diciembre 2011.

[Bioharness Data Sheet 11] "*Bioharness 3 Data Sheet*". Zephyr Technology. [En línea]. Noviembre 2011.

[Bioharness Specification 11] "*Bioharness Bluetooth Comms Link Specification*". Zephyr Technology. [En línea]. Noviembre 2011.

[Bioharness User Manual 11] "*Bioharness Bluetooth Developer Kit User Manual*". Zephyr Technology. [En línea]. Diciembre 2011.

[Bluegiga 12] Página web de la empresa Bluegiga. [En línea]. http://www.bluegiga.com/home. Agosto 2012.

[Bluetooth 12] Página web de la tecnología Bluetooth. [En línea]. http://www.bluetooth.com/Pages/History-of-Bluetooth.aspx. Agosto 2012.

[Bluetooth Especificaciones 12] Página web de Wikipedia sobre las especificaciones Bluetooth. [En línea]. http://es.wikipedia.org/wiki/Bluetooth#Especificaciones_y_novedades. Agosto 2012.

[Bluetooth Perfiles 12] Página web sobre los perfiles Bluetooth. [En línea]. https://www.bluetooth.org/Building/HowTechnologyWorks/ProfilesAndProtocols/Overview.htm. Agosto 2012.

[Bluetooth SIG 12] Página web de Wikipedia sobre la historia del SIG. [En línea]. http://es.wikipedia.org/wiki/Bluetooth_Special_Interest_Group. Agosto 2012.

[Brazier&09] Frances M.T. Brazier, Jeffrey O.Kephart, H. Van Dyke Parunak and Michael N. Hunhns. "*Agents and Service-Oriented Computing for Autonomic Computing*". IEEE Internet Computing. [Artículo]. 2009.

[Decker&00] Stefan Decker, Sergey Melnik, Frank Van Harmelen, Dieter Fensel, Michel Klein, Jeen Broekstra, Michael Erdmann e Ian Horrocks. "*The Semantic Web: The Roles of XML and RDF*". IEEE Internet Computing. [Artículo]. Octubre 2000.

[Fernández&09] Roberto Fernández Martínez, Joaquín Ordieres Meré, Francisco Javier Martínez de Pisón Ascacíbar, Ana González Marcos, Fernando Alba Elías, Rubén Lostado Lorza, Alpha Verónica Pernía Espinoza e Integrantes del grupo de investigación EDMANS. "*Redes inalámbricas de sensores: teoría y aplicación práctica*". Universidad de la Rioja. ISBN 978-84-692-3007-7. [Libro]. 2009.

[Hernando&08] Ana Belén García Hernando, José Fernán Martínez Ortega, Juan Manuel López Navarro, Aggeliki Prayati y Luis Redondo López. "*Problem Solving for Wireless Sensor Networks*". Ed Springer ISBN 978-1-84800-202-9. [Libro]. 2008.

[Hopkins&03] Bruce Hopkins and Ranjith Antony. "*Bluetooth for Java*". Ed. Apress. ISBN 1590590783. [Libro]. 2003.

[iWRAP SPP 10] "*Bluetooth serial port profile*". Versión 1.2. Bluegiga. [En línea]. Abril 2010.

[iWRAP4 12] "*iWRAP User Guide*". Version 4.1. Bluegiga. [En línea]. Enero 2012.

[Karl&07] Karl Holger and Willig Andreas. "*Protocols and Architectures for Wireless Sensor Networks*". Ed. Wiley. [Libro]. Págs. 17-32. 2007.

[LifeWear 12] Página web del Proyecto "LifeWear". [En línea]. http://www.lifewear.es/. Agosto 2012.

[LifeWear Deliverable 12] "*Deliverable 2.1 Platform requirements specification*". LifeWear. [Deliverable]. Julio 2012.

[LifeWear ESB 12] "*Enterprise Service Bus Proyecto LifeWear*". Julio 2012.

[Papazoglou&07] Michael P. Papazoglou, Paolo Traverso, Schahram Dustdar and Frank Leyman. **"***Service Oriented Computing: State of the Art and Research Challenges*". IEEE Computer Society. [Artículo]. Noviembre 2007.

[RFCOMM 03] "*RFCOMM with TS 07.10. Bluetooth Specification Version 1.1 (Part F: 1)*". [En línea]. Págs. 394 - 424. Junio 2003.

[Sparkfun 12] Página web del módulo Sparkfun WT32. [En línea]. https://www.sparkfun.com/products/8952. Agosto 2012.

[Specification 2.1 + EDR 07] "*Specification of the Bluetooth System. Covered Core Package version: 2.1 + EDR*". [En línea]. Julio 2007.

[SPP 01] "*Serial Port Profile. Bluetooth Specification Version 1.1 (Part K: 5)*". [En línea]. Págs. 172 - 196. Febrero 2001.

[SunSpot eDemo 10] "*SunSpot eDemo Technical Datasheet Rev 8.0*". Sun Oracle. [En línea]. Octubre 2010.

[WiMM 12] Página web de la empresa WiMM Labs. [En línea]. https://my.wimm.com/. Agosto 2012.

[WT12 Data Sheet 10] "*WT12 Evaluation Kit Data Sheet*". Bluegiga. [En línea]. Versión 1.7. Abril 2010.

[WT12 Product 09] "*WT12 Product Brief*". Bluegiga. [En línea]. Octubre 2009.

[WT32 Data 12] "*WT32 Data Sheet*". Bluegiga. Versión 2.12. [En línea]. Enero 2012.

[WT32 Product 09] "*WT32 Product Brief*". Bluegiga. [En línea]. Noviembre 2009.

[Zephyr 12] Página web de la empresa Zephyr. [En línea]. http://www.zephyr-technology.com/. Agosto 2012.